FPGA-BASED FILTERBANK IMPLEMENTATION FOR PARALLEL DIGITAL SIGNAL PROCESSING

Stephan Berner

NMSU-ECE-99-005 July 1999

FPGA-Based Filterbank Implementation for Parallel Digital Signal Processing

Stephan Berner

A technical report in partial fulfillment of the requirements for the Degree
Master of Science in Electrical Engineering

NMSU-ECE-99-005

Advisor: Prof. Phillip L. De Leon
New Mexico State University
Klipsch School of Electrical and Computer Engineering
Box 30001, Dept. 3-0
Las Cruces, New Mexico 88003

Table of Contents

1 Introduction....1
1.1 Subband processing, filter banks....1
1.2 Aliasing in the subbands....3
1.3 Polyphase representation for the critically sampled case....5
1.3.1 Some tools to derive the polyphase filter banks....5
1.3.2 Derivation of the polyphase filter bank....9
1.4 Polyphase representation for the oversampled case....13

2 Programmable logic....14
2.1 PLDs....14
2.2 FPGAs....15
2.3 Design entry....17

3 Implementation....18
3.1 Overview....18
3.2 Commutator....21
3.3 Upsampler....22
3.4 Filters....23
3.5 FFT/IFFT....24
3.6 Software tools....27
3.7 CLB counts....29

4 Testing the 4 subband analyzer/synthesizer filter banks....30
4.1 Sinusoidal input....30
4.2 Impulse response....33

5 Using DSPs for subband processing....35
5.1 Interfacing DSPs to the FPGAs....35
5.2 Operating the complete HW....38

6 Conclusion....40

7 References....41

Appendix

A MATLAB code for simulation
B VHDL code
C Impulse response, simulated with MATLAB
D Impulse response, simulated with ALDEC
E Motorola 56k code for pass through of subband signals
F Block diagram of hardware
G Schematics
H FPGA pin assignments

1 Introduction

1.1 Subband processing, filter banks

It may be desired to apply Digital Signal Processing (DSP) to high bandwidth signals with a sampling rate too high for being handled by any commercially available processor. One approach to solve this problem is to divide the spectrum of the signal into subbands by an analysis filter bank, then process the subbands in parallel, and finally, recombine the processed subband signals by a synthesis filter bank. The key point is the fact, that the sampling rate in the subbands can be reduced by downsampling, because the bandwidth of the subband signals is reduced. Figure 1.1 shows this idea. This report describes a hardware implementation of Figure 1.1. The analysis and synthesis filter banks are implemented with Field Programmable Gate Arrays (FPGAs), the subband processing is accomplished with Motorola DSP 56302 Evaluation Modules (EVMs).

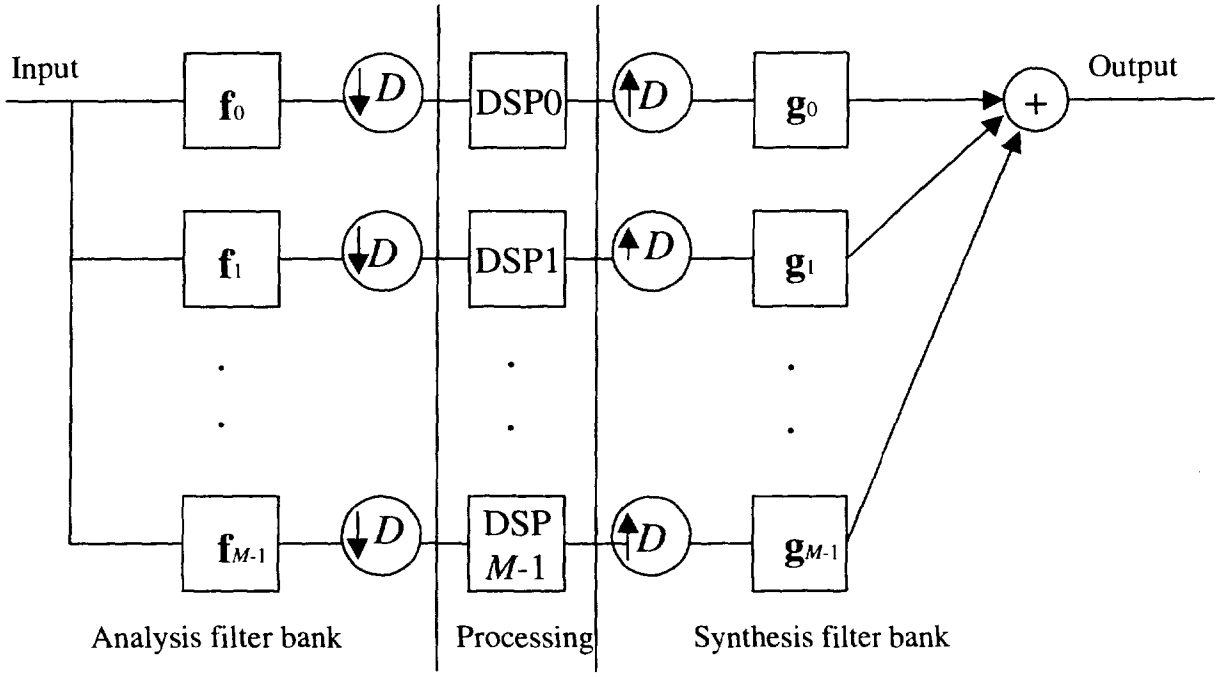


Figure 1.1: Parallel DSP

The analysis filter bank consists of a set of bandpass filters, usually non overlapping except near the crossover frequency of two subbands. Figure 1.2 shows a possible magnitude response of the bandpass filters. After the bandpass filters, the sampling rate can be reduced without aliasing according to the Nyquist theorem, because the bandwidth of the signal is reduced. The synthesis filter bank consists also of a set of bandpass filters, necessary to cancel the images, which are introduced during upsampling. For more filter bank details and theory of multirate systems see [2], [6].

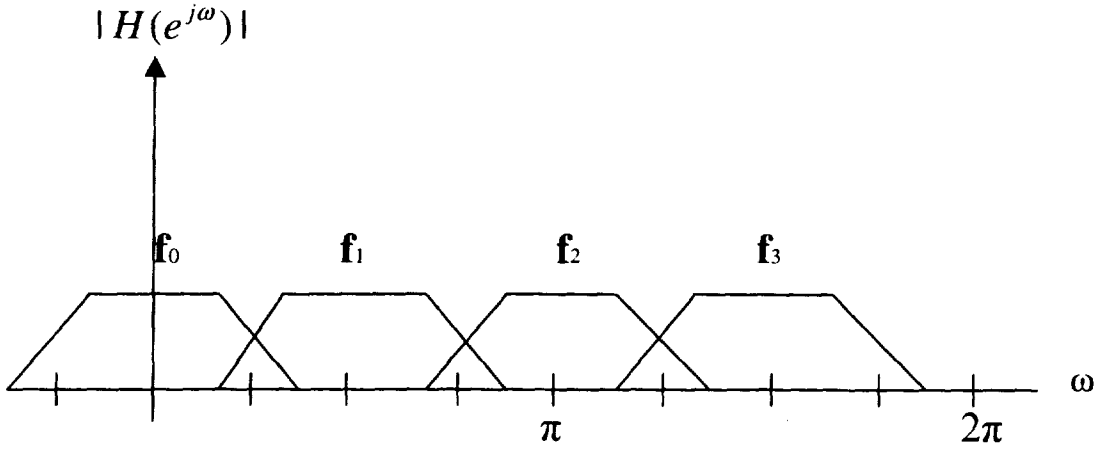


Figure 1.2: Frequency response of bandpass filters for the case of 4 subbands

Filter banks are also used for applications other than parallel processing. One important application is signal compression, which makes use of the fact, that the spectral energy of a signal is often concentrated in a certain band, e.g. around DC. Therefore, the subband signals, which represent the higher frequencies, can be encoded with less bits or even omitted completely. Another application is spectral analysis. Here, the subband signals are used to obtain information about the energy distribution in the signal. A desirable property of an analysis filter bank combined with a synthesis filter bank is perfect reconstruction. That means, the output signal, $\hat{x}[n]$ of the synthesis bank is a scaled and delayed version of the input signal, $x[n]$.

$$\hat{x}[n] = cx[n - n_0]$$

The major issue is the aliasing, caused by the non ideal filters. One approach to this problem is to construct the filters in the synthesis bank such that the aliasing, introduced in the subbands is cancelled [2], [6]. This solution works fine as long as there is no processing performed in the subbands. Another approach is oversampling, that means the sampling rate in the subbands is not as much reduced as it is theoretically possible according to the Nyquist theorem. More details can be found in the next section. An important class of filter banks is the uniform DFT filter bank. The bandpass filters of this filter bank are derived from a single lowpass filter by modulation (frequency shift).

$$f_m[n] = f_0[n]e^{j\frac{2\pi mn}{M}}$$

Therefore, all subband signals have the same bandwidth. In this work, only uniform DFT filter banks are used.

1.2 Aliasing in the subbands

This section shows the aliasing in the subbands, and how it can be prevented by oversampling. Equation (1) gives the spectrum of a downsampled signal, with DTFT, .

$$X_d(e^{j\omega}) = \frac{1}{D}\sum_{k=0}^{k=D-1} X(e^{j(\frac{w-2\pi k}{D})}) \qquad (1)$$

The spectrum of the downsampled signal is a sum of the expanded original spectrum and several shifted copies. Figure 1.3 shows the spectrum of one subband of the Uniform DFT filter bank for the critically sampled case, that means the downsample factor is equal to the number of subbands. Four subbands and nonideal filters are assumed. Figure 1.4 shows the oversampled case, an oversampling factor of 2 is assumed.

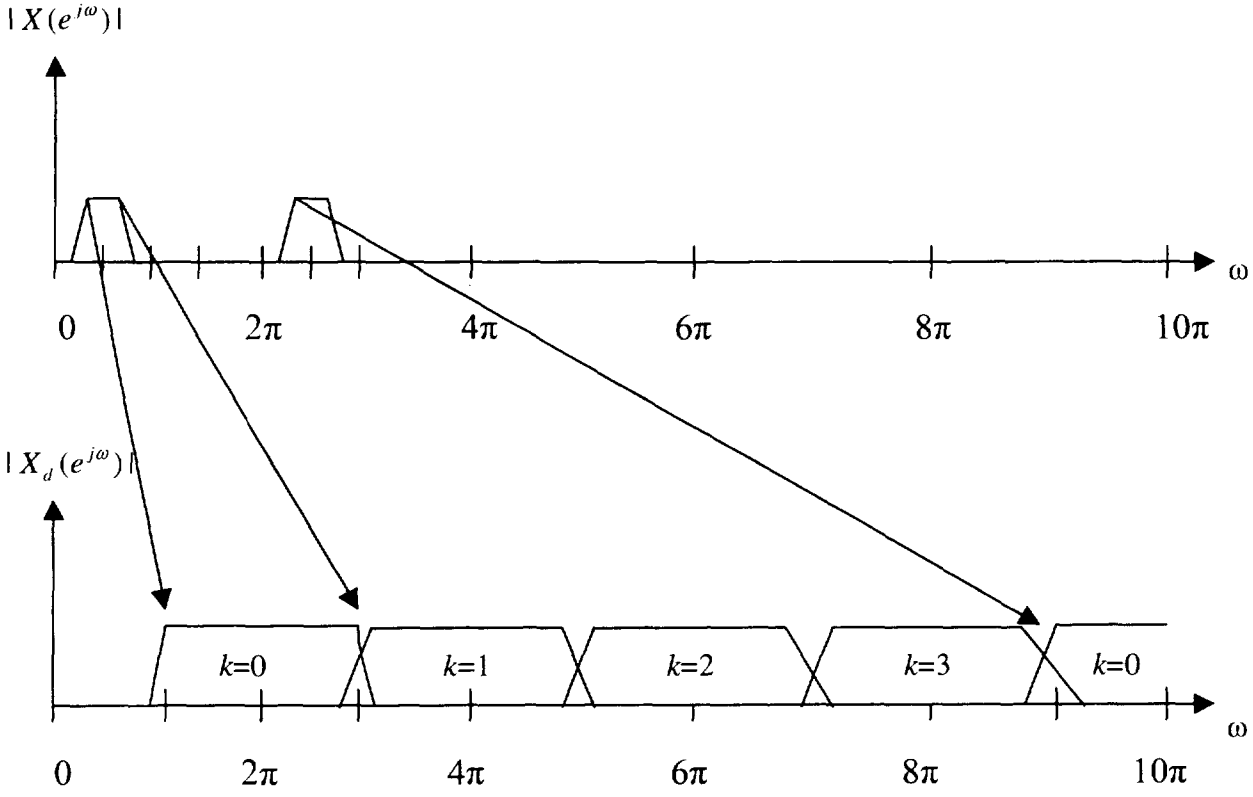


Figure 1.3: Subband spectrum for the critically sampled case

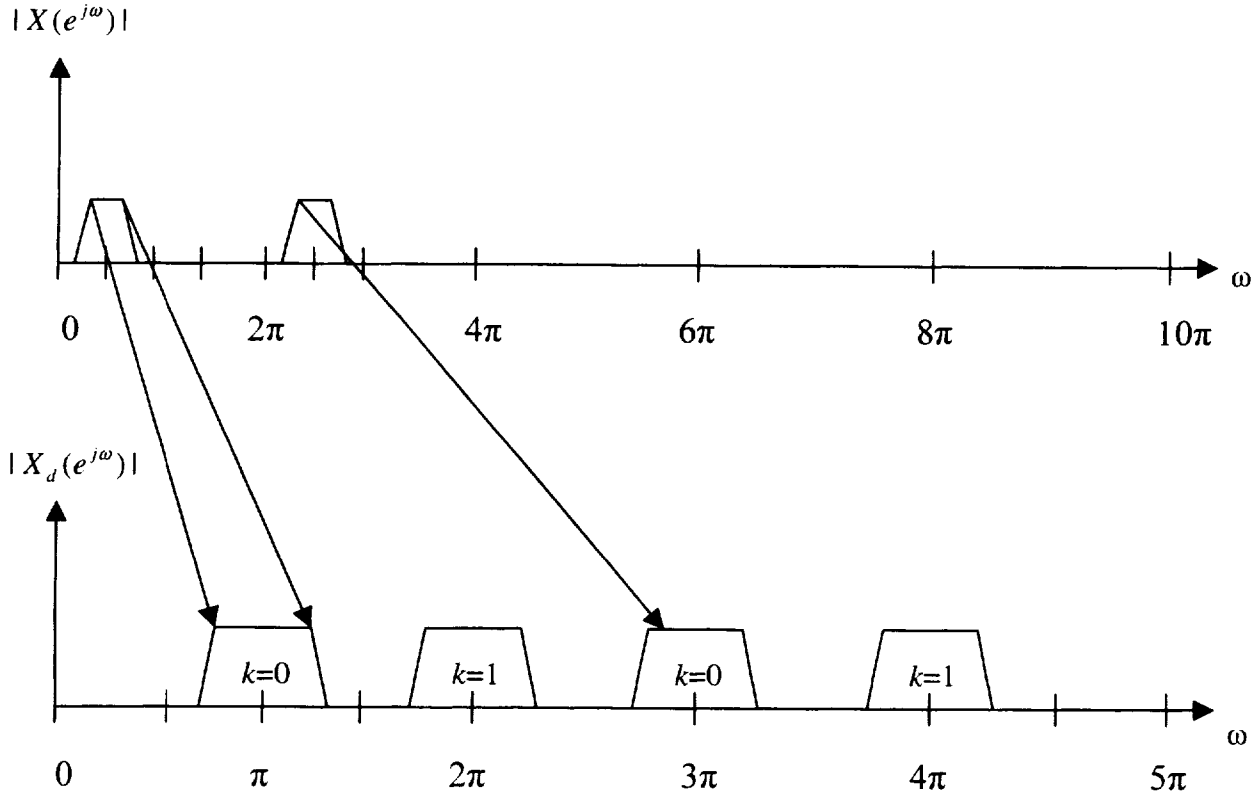


Figure 1.4: Subband spectrum for the 2 times oversampled case

It can be seen, that for the critically downsampled case in Figure 1.3, there is an overlapping (aliasing) of the expanded copies of the original spectrum, while for the oversampled case in Figure 1.4, there is a gap between the expanded copies of the original spectrum.

1.3 Polyphase representation for the critically sampled case

The implementation of the filter banks given in Figure 1.1 is straightforward, but inefficient regarding the usage of hardware resources. The filters of the analysis bank are running at the high input sample rate. After the downsamplers, many samples, calculated previously, are thrown away. Furthermore, the bandpass filters have complex coefficient, which also complicates the implementation. A more efficient implementation, equivalent to Figure 1.1 is given in Figure 1.5. Section 1.3.1 gives some tools to transform digital networks, section 1.3.2 derives the polyphase filter bank of Figure 1.5 from the direct form in Figure 1.1.

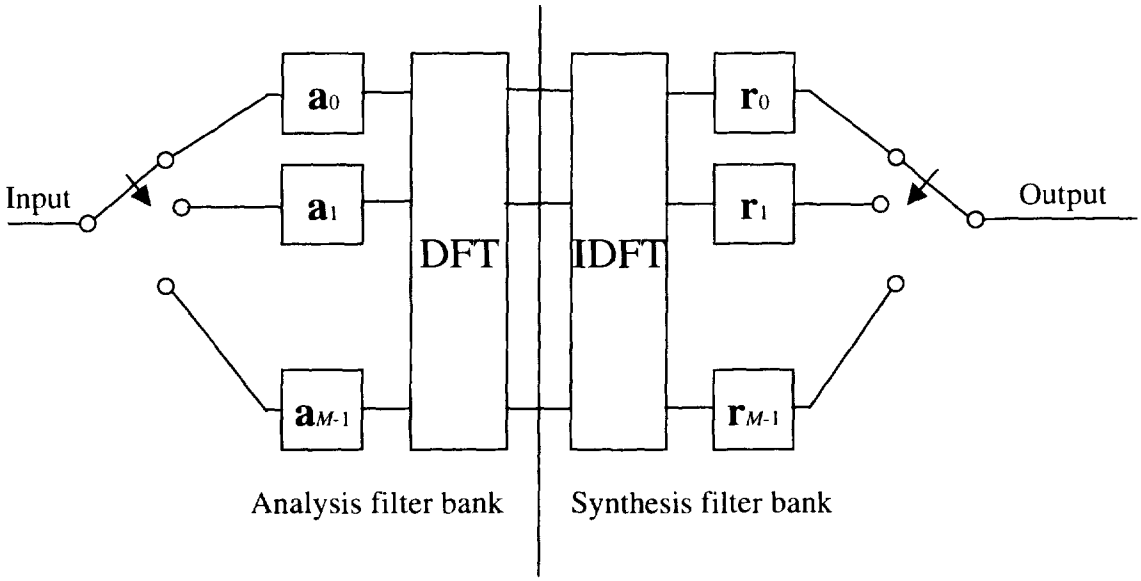


Figure 1.5: Polyphase filter banks for the critically sampled case

1.3.1 Some tools to derive the polyphase filter banks

This section gives some useful tools to derive the polyphase form of the filter bank, given in Figure 1.5, from the direct form, given in Figure 1.1.

Polyphase representation:
Every sequence, a signal as well as an impulse response of a filter, can be represented as a sum of upsampled and shifted subsequences (the polyphase sequences).
Figure 1.6 gives an example.

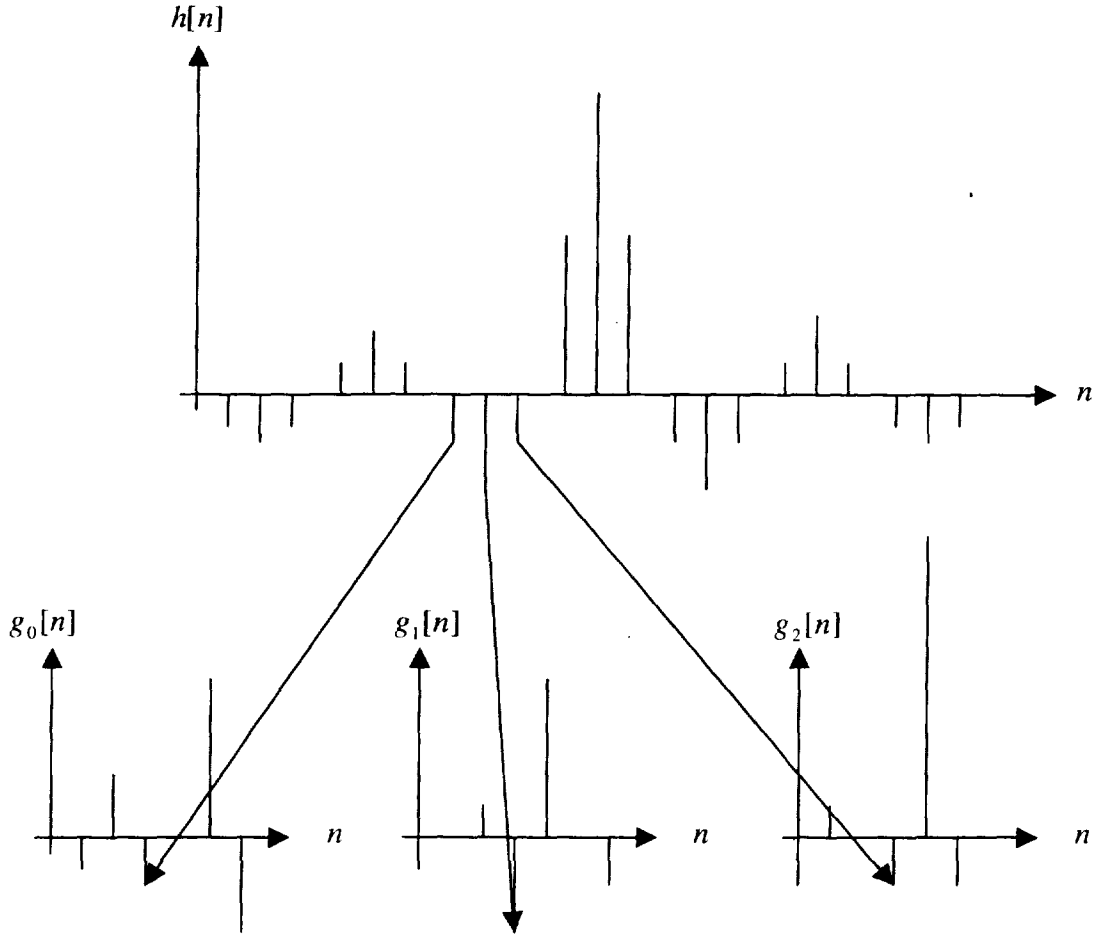


Figure 1.6: Decomposition of a sequence into 3 polyphase sequences

Equation (2) gives the polyphase representation of $h[n]$, M=3.

$$h[n] = \sum_{k=0}^{M-1} g_k[\frac{n-k}{M}] \quad , \tag{2}$$

where

$$g_k[\frac{n-k}{M}] = \begin{cases} \frac{n-k}{M} & \text{if } n\text{-}k \bmod M = 0 \\ 0 & \text{otherwise} \end{cases}$$

The z transform of equation (2) yields equation (3).

$$H(z) = \sum_{k=0}^{M-1} z^{-k} G_k(z^M) \tag{3}$$

Nobel identities:
Figure 1.7 shows the Nobel identities, which allow to interchange sample rate conversions and filters.

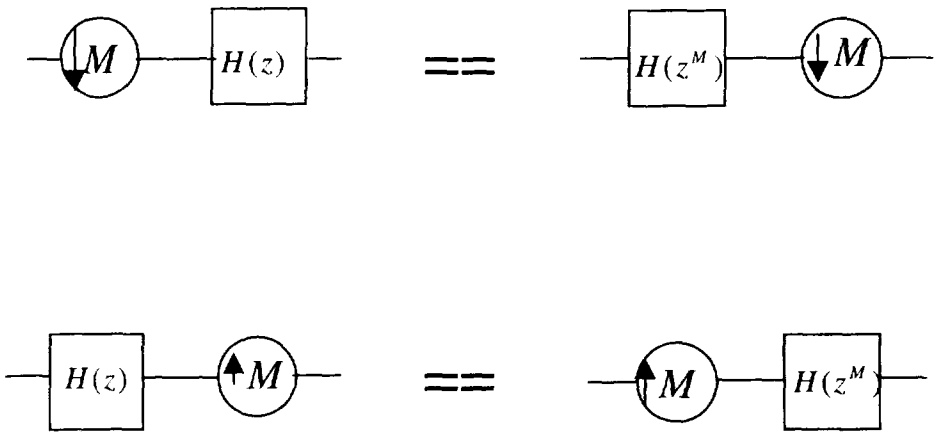


Figure 1.7: Nobel identities

The polyphase representation leads to significant computational savings, if applied to a filter followed by a downsampler. Figure 1.8 shows this.

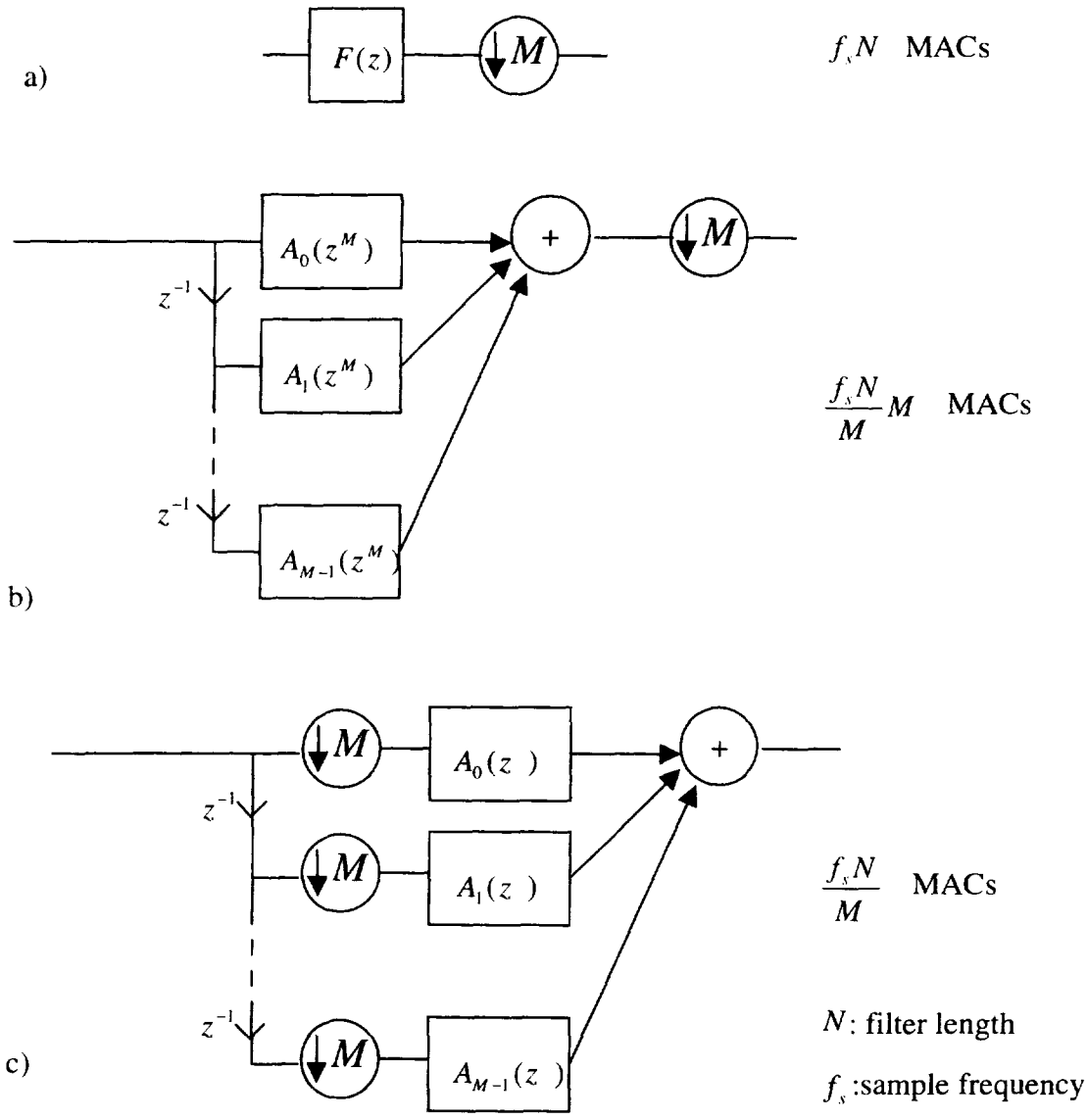


Figure 1.8: Computational savings by using the polyphase form

Figure 1.8 a) shows some filter followed by a downsampler, in Figure 1.8 b), the filter is replaced by its polyphase representation. This alone does not lead to any computational saving. After moving the downsampler, using the first Nobel identity, the number of computations is reduced by a factor of M (Figure 1.8 c)).

1.3.2 Derivation of the critically sampled polyphase filter bank

As a first step, each filter in Figure 1.1 can be replaced by its polyphase form. Figure 1.9 shows the result for the analysis bank.

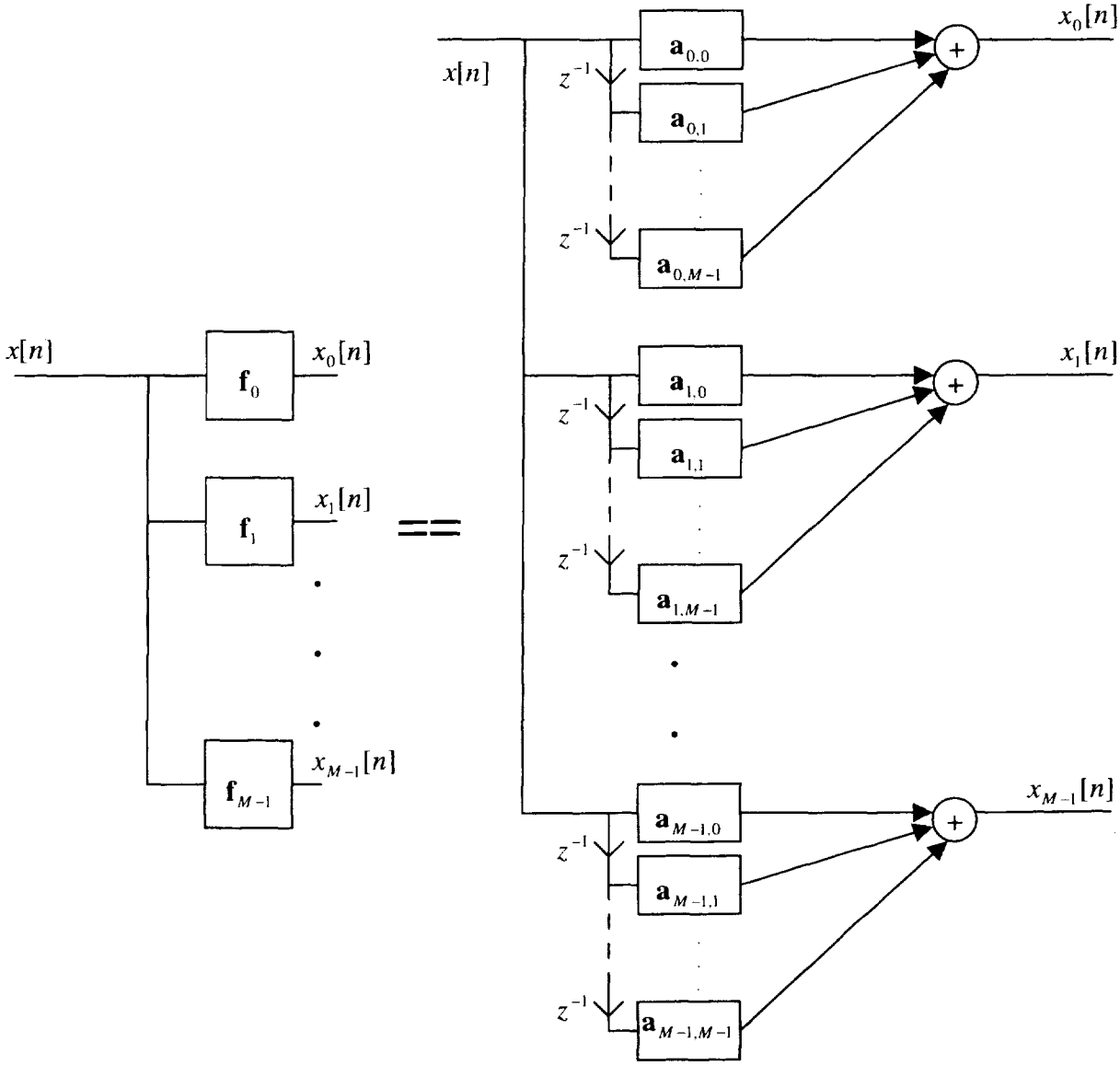


Figure 1.9: Filters of analysis bank replaced by polyphase form

This step alone reduces, after moving the downsamplers, the number of computations by a factor of M. Further reduction can be achieved for an important special case, where the bandpass filters are derived from one lowpass by frequency shifting, i.e. uniform DFT filter bank.

The polyphase form of the lowpass is given by

$$F_0(z) = \sum_{k=0}^{M-1} z^{-k} A_k(z^M) \quad . \tag{4}$$

The bandpass filters are a frequency shifted version of (4).
With

$$W_m = e^{j\frac{2\pi}{M}} \quad ,$$

$$F_m(z) = F_0(ze^{-j\frac{2\pi m}{M}}) \quad . \tag{5}$$

We note the Z transform property

$$z_0 x[n] \leftrightarrow X(\frac{z}{z_0}) \quad . \tag{6}$$

The polyphase form of the bandpass filters is derived by substituting (4) into (5) and applying (6).

$$F_m(z) = \sum_{k=0}^{M-1} z^{-k} W_M^{mk} A_k(z^M) \tag{7}$$

The set of filters, obtained by taking equation (7) for $0 \le m \le M-1$, can be written in matrix form.

$$\begin{bmatrix} F_0(z) \\ F_1(z) \\ . \\ . \\ F_{M-1}(z) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & . & W^0 \\ W^0 & W^1 & . & . & W^{M-1} \\ W^0 & . & W^4 & . & . \\ . & . & . & . & . \\ W^0 & W^{M-1} & W^{2(M-1)} & . & W^{(M-1)^2} \end{bmatrix} \begin{bmatrix} z^{-0}A_0(z^M) \\ z^{-1}A_1(z^M) \\ . \\ . \\ z^{M-1}A_{M-1}(z^M) \end{bmatrix} \quad (8)$$

The matrix is recognized as the DFT matrix. Therefore, the following implementation given in Figure 1.10 is possible, if the filters are derived from one common lowpass filter by frequency shift.

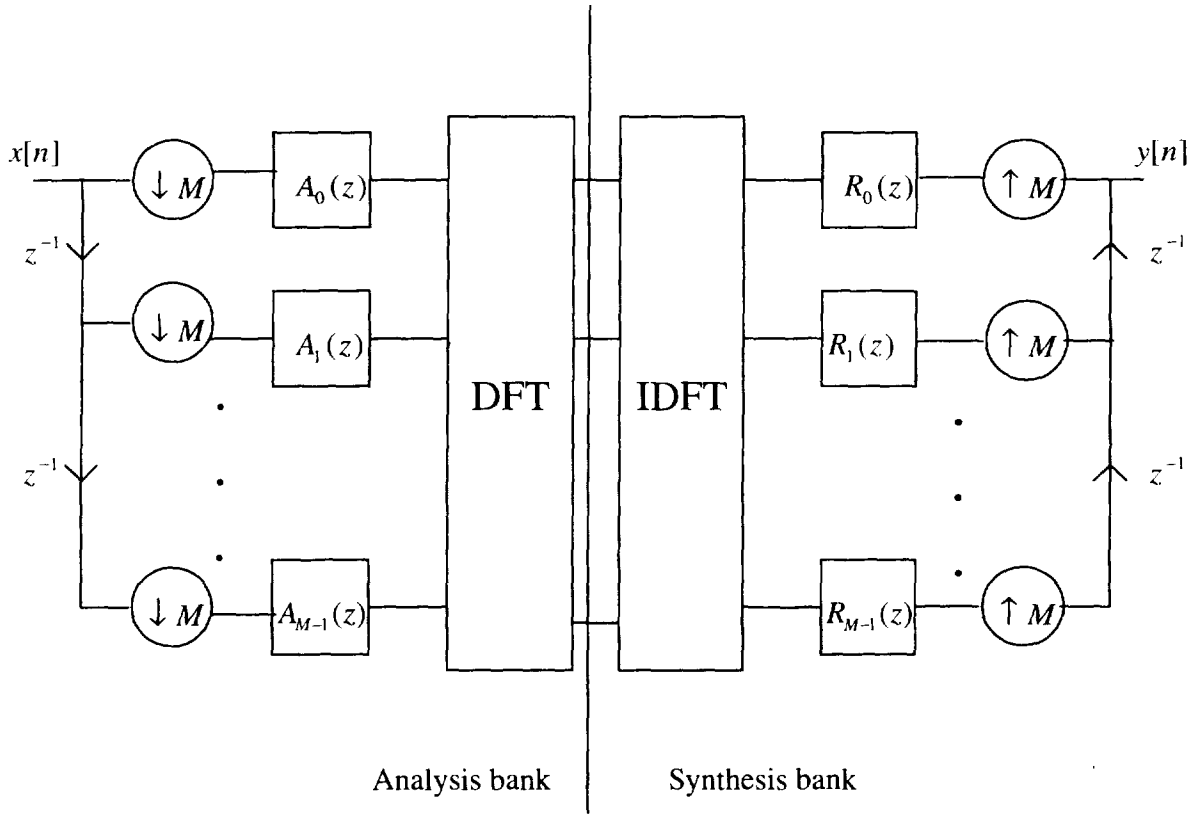


Figure 1.10: Implementation of uniform DFT filter bank using polyphase form

Finally, the combination of downsamplers and delays can be replaced by a commutator as shown in Figure 1.11.

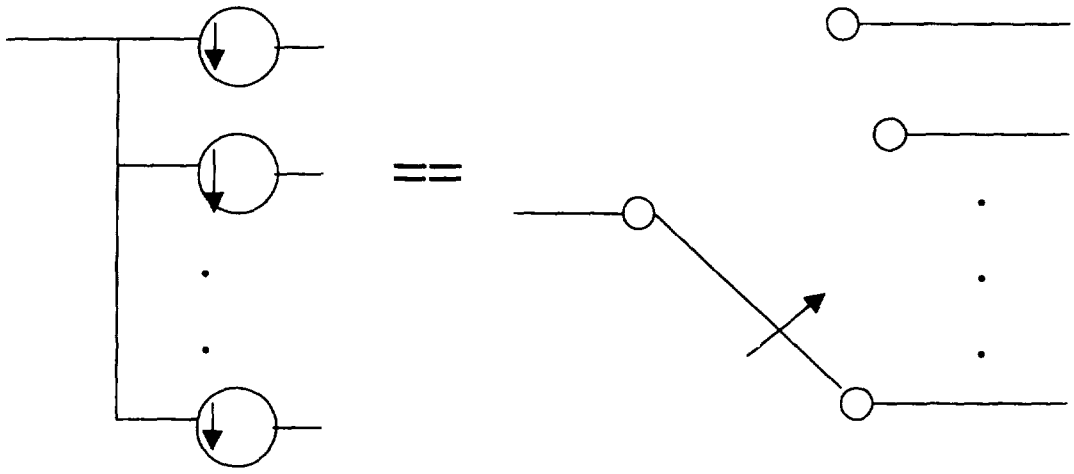


Figure 1.11: Delays and downsampler replaced by commutator

After these steps, the implementation of Figure 1.5 is obtained.

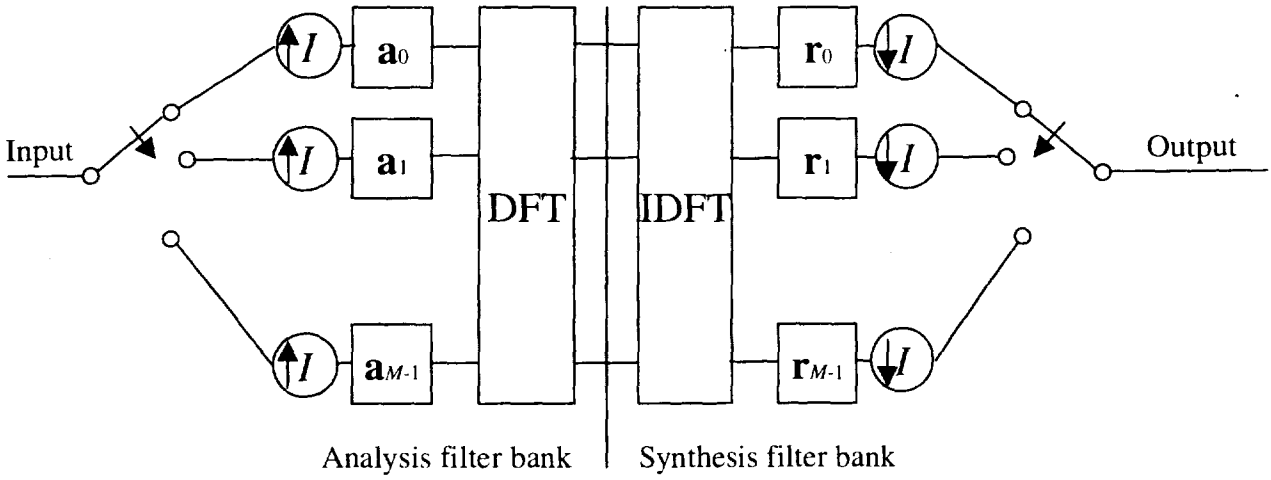
1.4 Polyphase representation for the oversampled case

The polyphase filter bank for the oversampled case is shown in Figure 1.12. The derivation is not as straightforward as for the critically sampled case. A derivation can be found in [2], page 311. The method is to write the subband signals as convolution sums, and then manipulate these sums in order to get something like a polyphase representation. The polyphase filters for the analysis bank with down-sampling factor D, M subbands, assuming a clockwise commutator, can be obtained from the lowpass prototype $f_0[n]$ with (9) as

$$a_m[n] = f_0[nD - m] \quad , \quad 0 \le m \le M - 1 \; . \tag{9}$$

The polyphase filters for the synthesis bank with upsampling factor D, M subbands, assuming a counterclockwise commutator, can be obtained from the lowpass prototype $g_0[n]$ with (10) as

$$r_m[n] = g_0[nD + m] \quad , \quad 0 \le m \le M - 1 \; . \tag{10}$$



$I = \frac{M}{D}$: Oversampling factor , M: Number of subbands, D: Downsampling factor

Figure 1.12: Polyphase filter banks, I times oversampled

Here, for this project, the oversampling factor I is set to 2.

2 Programmable Logic

For prototyping a large digital system, like the analysis and synthesis filter bank, programmable logic is the only reasonable choice, because of hardware size, flexibility to change the function, time to build the circuit (wire wrap), ... associated with the use of Small Scale Integration (SSI) devices like TTL, ECL,.... This chapter gives a short overview of programmable logic. There are 2 major groups of programmable logic, Programmable Logic Devices (PLDs), better known as Programmable Array Logic (PAL), and Field Programmable Gate Arrays (FPGAs).

2.1 (Programmable Logic Devices) PLDs

Every logic function can be expressed as a "sum of product terms", for example

$$y = x\ x_2\overline{x_3} + x_1\overline{x_2} + \ldots.,$$

where the product means "logic and", and the summation means "logic or".
The PAL architecture is based on this fact. Figure 2.1 shows the basic components, the details depend on the manufacturer and can be found in the data books (example AMD, Lattice, Xilinx,....).

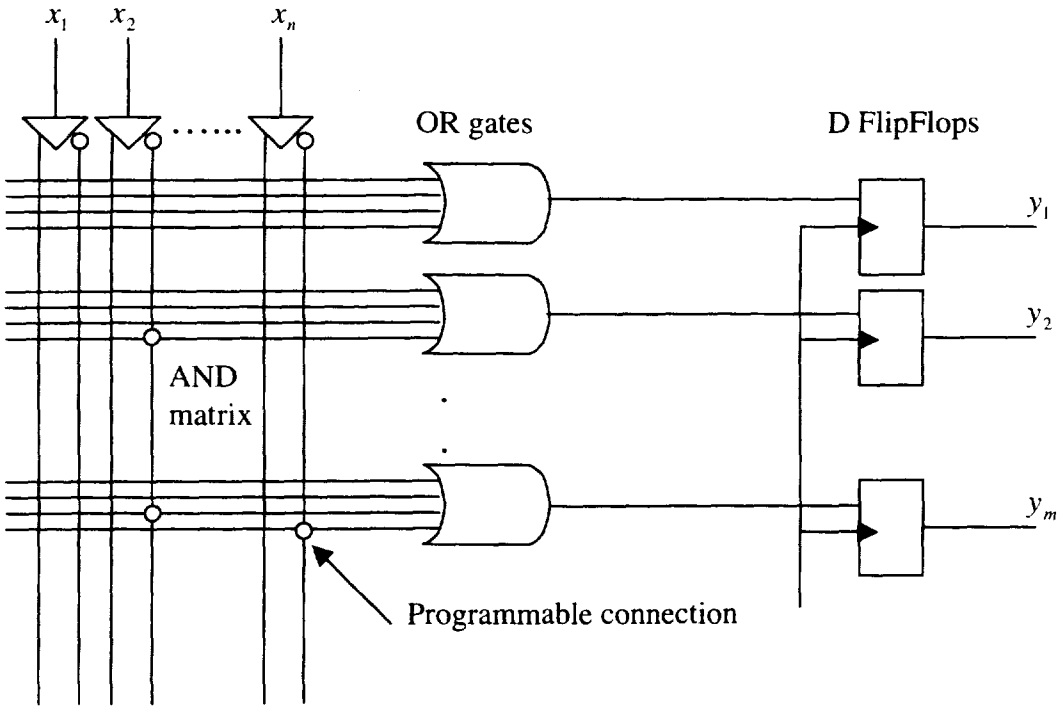


Figure 2.1: PAL architecture

The product terms are determined by the connections in the AND matrix, which can be reversible (like an EPROM or EEPROM) or permanent (fuse technology), depending on type and manufacturer. PALs are well suited for smaller circuits without many pipeline stages, e.g. address decoder. Larger, pipelined circuits require passing output signals again through the switch matrix, which is likely to cause congestion in the switch matrix (AND matrix).

2.2 Field Programmable Gate Arrays (FPGAs)

The architecture of FPGAs is completely different, making them more suitable for larger, pipelined circuits. A FPGA consists of a large number of Configurable Logic Blocks (CLBs), each CLB includes one or more FlipFlops and some logic. The CLBs can be connected by switches. The state of the switches and the configuration of the CLBs is determined by configuration data, which is downloaded from a computer to the FPGA, or which can be read from an EPROM by the FPGA itself. Figure 1.14 shows the basic FPGA architecture, more details can be found in the Xilinx data book [7] , page 4-1 to 4-40.

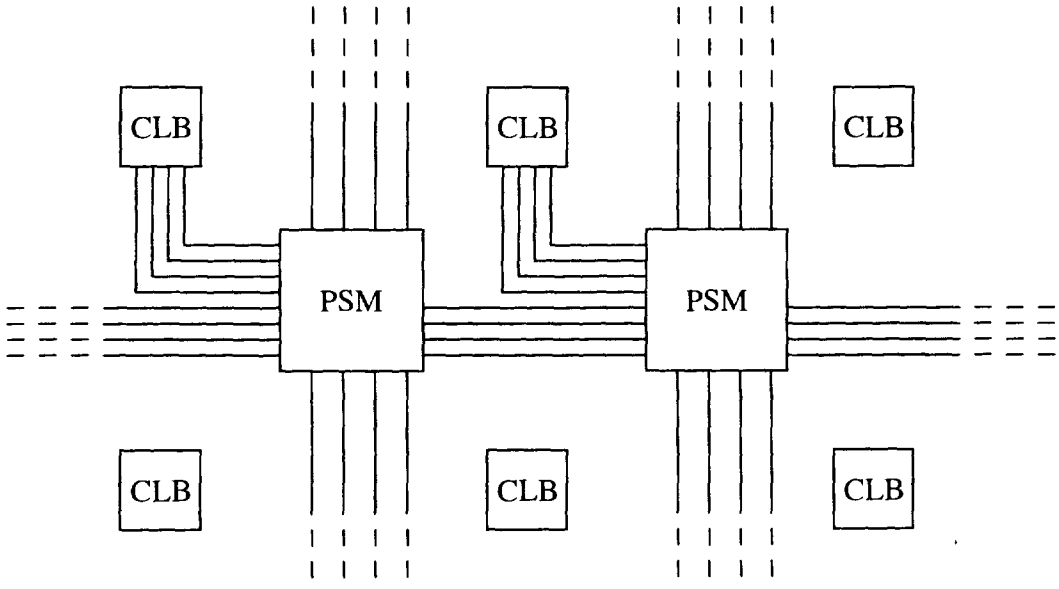


Figure 2.2: FPGA architecture

The CLBs can be connected by the Programmable Switch Matrices (PSMs) in an arbitrary way, which allows the implementation of different functions in the FPGA. The PSMs consist of transistors, which connect or disconnect signal lines. The state of the switches is stored in SRAM somewhere in the FPGA after loading configuration data from a PC or from an EPROM. Figure 2.3 shows the basic idea.

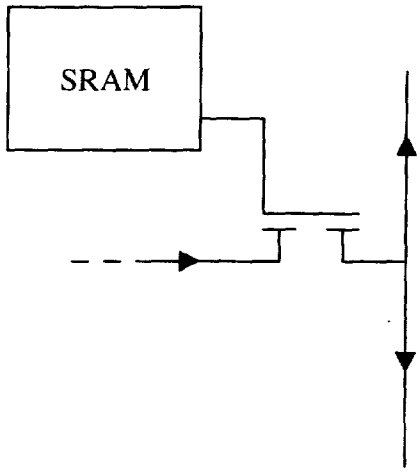


Figure 2.3: Principle of the PSM

Figure 2.4 shows the basic parts of a CLB. Details can be found in the Xilinx data book [7], 4-1 to 4-30. The logic functions are implemented with a Lookup table (LUT), that means the input signals represent the address of a RAM, and the stored data depends from the desired logic function. The data, which defines the logic function, is loaded during configuration of the FPGA. It can be seen, that a number of cascaded CLBs is well suited to build pipelined digital systems.

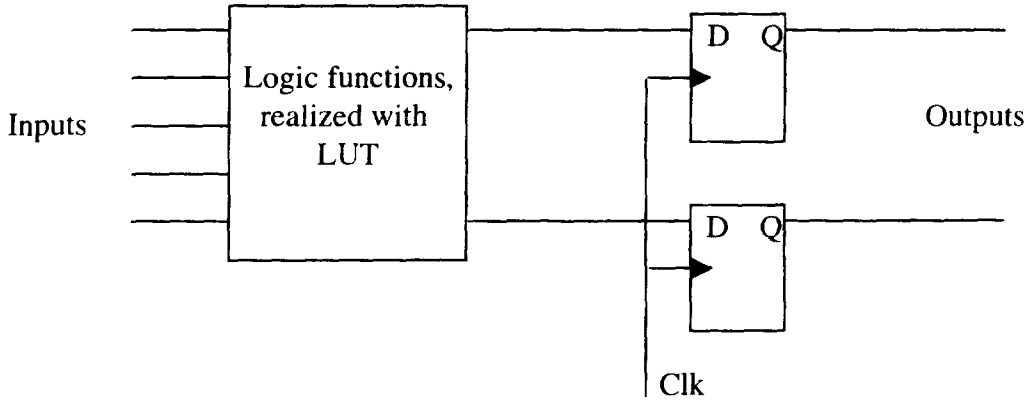


Figure 2.4: CLB

2.3 Design entry

Schematics:
The user enters a schematic diagram, which is translated into configuration data for the FPGA or PAL by the design software. This method may be easier to learn than a hardware description in text form, it also provides a good documentation (the schematic) during design entry, but it does not provide the flexibility of text oriented methods, and it often leads to less efficient implementations. Especially, it is not recommended to use common TTL ICs (which is possible with some design software) as components during schematic entry. According to the author's experience, this leads usually to very inefficient implementations.

State diagram:
The user enters a state diagram, the design software translates it to configuration data. The advantage is the good documentation (the state diagram), which is created during design entry, otherwise this method is equivalent to a state machine description in text form.

Text oriented methods:
Examples are Very High Speed Integrated Circuit Hardware Description Language (VHDL), PALASM from AMD,.... VHDL is more abstract compared to other ways of design entry like schematics, but it provides much more flexibility. For example, it is easy to make a design scaleable. The subband data is represented by a 2 dimensional array in VHDL, one dimension for the number of subbands (4, 8, 16,...), the other dimension for the number of bits per subband. By changing the dimensions of the array, the design can be scaled easily. Another feature of VHDL is the conditional generation of logic dependent on some condition like the number of subbands,.... VHDL code looks like program source code, but it describes hardware rather than instructions for a CPU.

3 Implementation

3.1 Overview

This chapter describes the implementation of the analyzer and synthesizer filter bank in VHDL code. Figure 3.1 shows again a block diagram of the whole system, Figure 3.2 shows the VHDL files created.

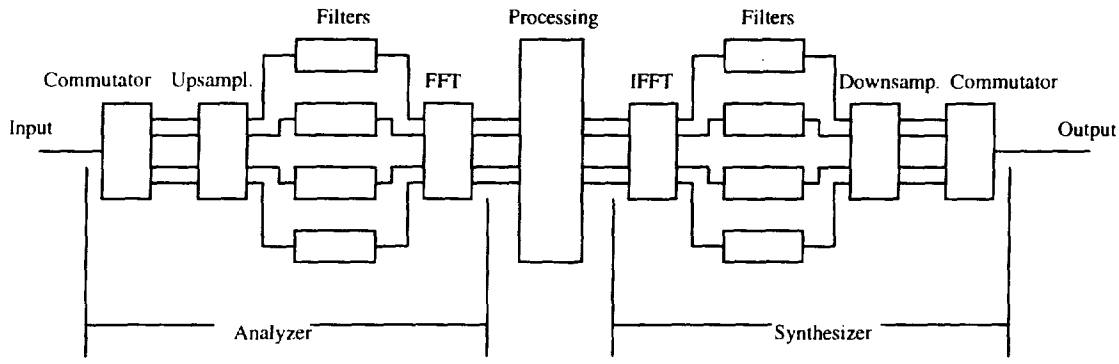


Figure 3.1: Blockdiagram of system for 4 subbands

The number of subbands can be set to 4, 8 or 16 by modifying line 12 in the file Fltb_par.vhd, App. B, which includes constants and type definitions similar to a header file in C. An extension to 32, 64,... subbands requires some slight modifications, because for 4, 8, 16 subbands the clock of the filters is the fastest clock in the system, but for 32, 64,... subbands the sample clock (the input data rate) becomes the fastest clock. The word sizes can also modified easily in the file Fltb_par.vhd, which allows it to trade precision against consumption of hardware. Table 3.1 shows the data rates in the system for different numbers of subbands assuming a system clock of 64 MHz. This clock of 64 MHz is an optimistic assumption, with Xilinx XC 4025E -3 devices, a maximum clock of about 25 MHz has been achieved. It may be possible to increase it by timing optimization and using a faster device (-3 is a slow device). Fb_n.vhd, App. B, is the highest level file, it contains 2 components, the analysis and synthesis banks, which are described in the files an_n.vhd and sy_n.vhd, App. B. The analysis bank contains the components commutator, upsampler, filter, fft, which are described in lower level files, and so on.

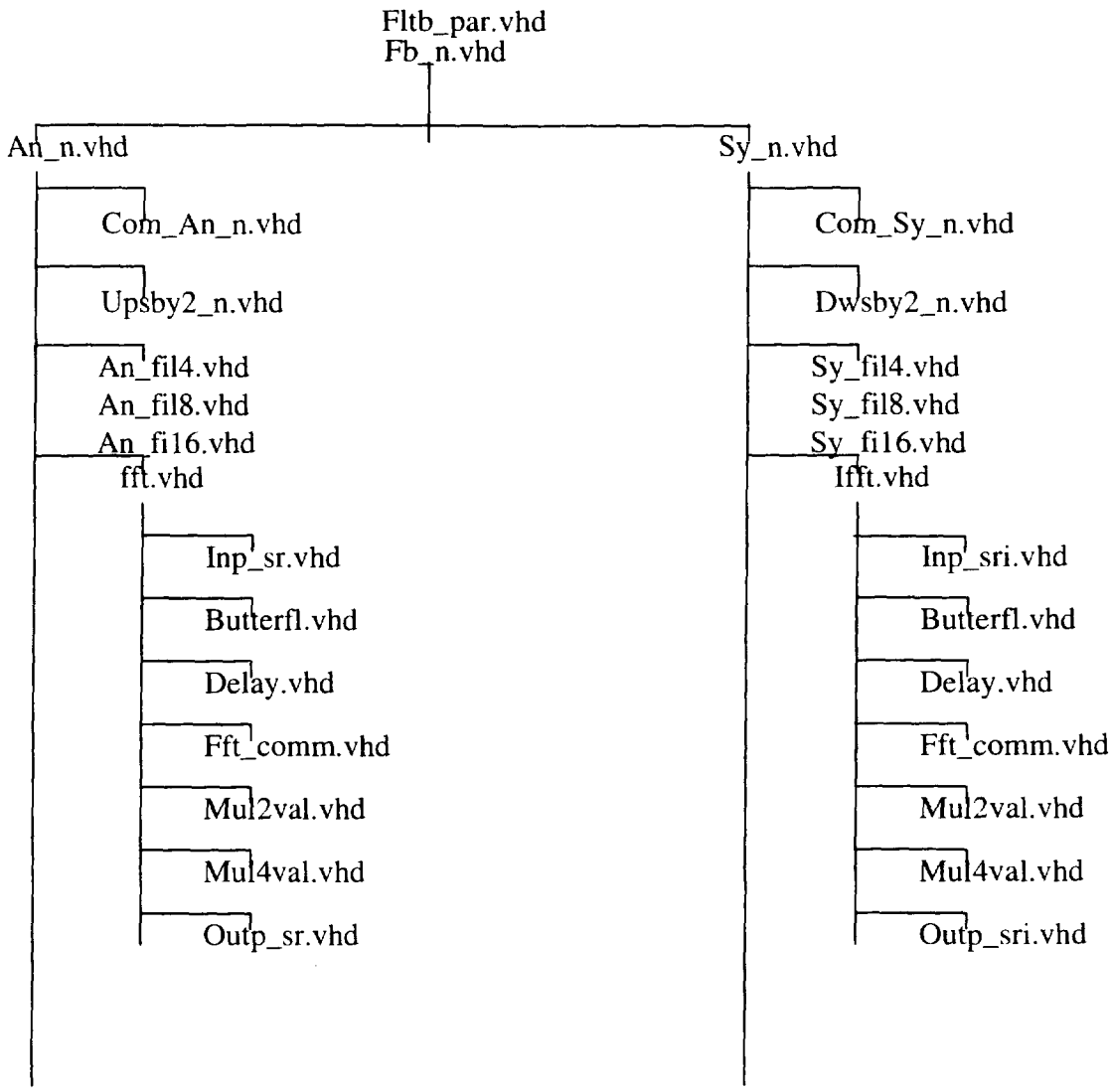


Figure 3.2: Overview of VHDL files composing analysis and synthesis banks

The constant "COMM_OFFSET" in the file fltb_par.vhd, line 52 determines the position of the commutator of the synthesis bank relative to the commutator of the analysis bank, which has to be adjusted due to additional delays not considered in the theory [2]. The adjustment is done by simulation using the Aldec simulator. If the adjustment is wrong, the samples in the output sequence are not in the right order, a good input signal to do this test is a ramp. Another related issue is the offset of the downsampler, e.g. which samples should be thrown away, but this is solved in another way. A jumper allows to shift the downsampler of the synthesis bank easily by 1 sample, see Appendix H, Xilinx Pad spec. file for Sy4, downsampl_del.

Table 3.1: Datarate at different points of the system

	datarates/word lengths		
Number of subbands	4	8	16
Input sampling rate	16 Mhz /8 bit	32 Mhz /8 bit	64 Mhz/8bit
Commutator output	4 Mhz/8 bit	4 Mhz /8 bit	4 Mhz/8 bit
Upsampler output	8 Mhz/8 bit	8 Mhz/8 bit	8 Mhz/8 bit
Filter output	8 Mhz/21 bit	8 Mhz/21 bit	8 Mhz/21 bit
FFT output	8 Mhz/22 bit	8 Mhz/23 bit	8 Mhz/24 bit
Analyzer output	8 Mhz/23 bit	8 Mhz/24 bit	8 Mhz/25 bit
Synthesizer input	8 MHz/8 bit	8 MHz/8 bit	8 Mhz/8bit
IFFT output	8 Mhz /10 bit	8 Mhz /11 bit	8 Mhz/12bit
Filter output	8 Mhz/21 bit	8 Mhz /21 bit	8 Mhz/21 bit
Downsampler output	4 Mhz/21 bit	4 Mhz/21 bit	4 Mhz/21 bit
Commutator output	16 Mhz/21 bit	32 Mhz/21 bit	64 Mhz/21 bit
Output sampling rate	16 Mhz/21 bit	32 Mhz/21 bit	64 Mhz/21 bit

3.2 Commutator

Figure 3.3 shows a schematic of one possible realization of the commutator. The shift register includes only one "one", the other bits are cleared. The "one" rotates around and enables one of the data registers, which is updated with the current sample.

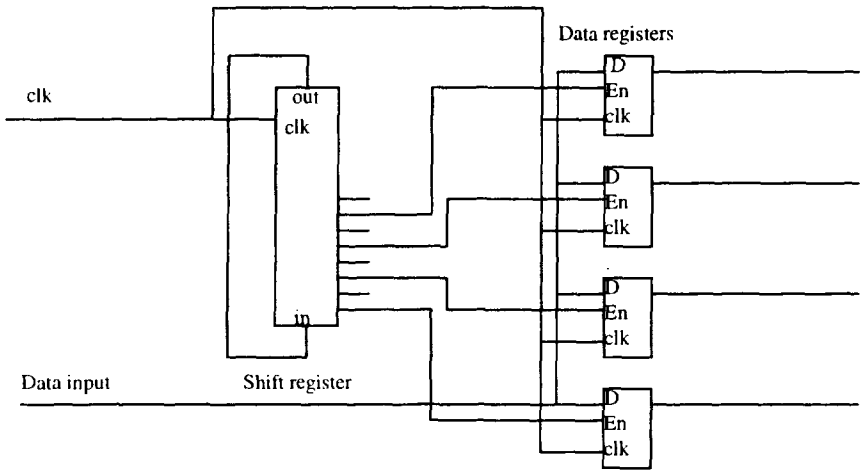


Fig.3.3: Commutator for filterbank with four subbbands

The file "Com_An_n.vhd" in App. B shows the equivalent VHDL code. The "entity" section, line 14, defines the interface of the module "Com_An" to other modules, the section "architecture" , line 22, defines the actual implementation of "Com_An". All logic is included in a "process'. The first process, line 29, implements the shift register. The shift register is initialized with 00000...000001 if "reset" is low, otherwise the content is rotated by one bit with every rising clock edge. The second process, line 39, implements the output registers, the "for loop" builds as many registers as needed for the given number of subbands. The third process, line 52, makes just some signal connections.

3.3 Upsampler

Figure 3.4 shows a schematic of the upsampler, the corresponding VHDL code can be seen in Appendix B. The multiplexer inserts a "zero" for every other sample.

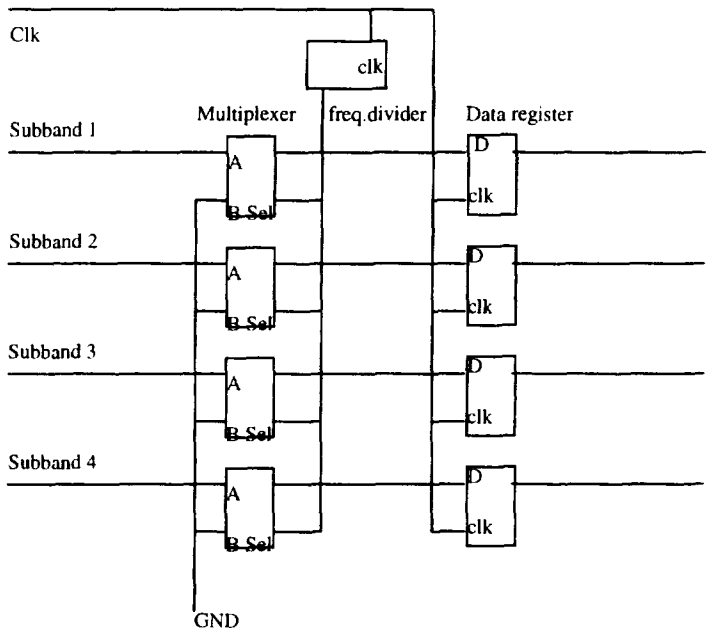


Figure 3.4: Upsampler for filterbank with four subbands

3.4 Filters

As a first step, the polyphase filters are generated from a lowpass prototype using the Matlab program provided in Appendix A. The polyphase filters are obtained from a lowpass prototype by downsampling and shifting its impulse response. The cutoff frequency of this lowpass is π/M (M number of subbands) for both, critically downsampled and oversampled, types of filter banks. In this project, the lowpass prototype is designed with a rectangle window for simplicity, which leads to the poorest performance regarding the suppression of side lobes in the impulse response compared to other design methods. The program calculates the coefficients for the analyzer filters, for the synthesizer filters, it normalizes the coefficients to 8 bit, it calculates and plots also the impulse response of the combination analysis/synthesis bank considering a data truncation in the subbands.

The second step is to generate the filters with Core Generator, a software tool provided by Xilinx. The Core Generator allows the generation of scaleable and optimized high level components like filters, adders, multipliers,.... There are two types of FIR filters available: Serial Distributed Arithmetic (SDA) and Parallel Distributed Arithmetic (PDA). SDA and PDA are well described in a Xilinx application note, page 6 to 10, [10]. Filters using SDA are efficient regarding the usage of hardware, but allow only moderate throughput rates, while PDA allows a higher throughput using more hardware. For example, an 32 tap filter, 8 bit data and coefficient size requires 107 CLBs using SDA, the maximum throughput is 8.9 Mhz. The same filter realized with PDA would require about 740 CLBs with a maximum throughput of 80 Mhz. These estimates are based on the Core Generator data sheets, which are included in the Core Generator documentation ("help"). Here SDA filters are suitable because PDA filters would become prohibitively large, and a subband frequency above 8.9 Mhz, which is the the maximum speed for SDA filters, is not desired at this time. The subband datarate is equal to the maximum datarate of the filters, the input datarate of the analysis bank is coupled to the subband datarate according to Table 3.1. Core Generator generates a *.edn file, which is used by the Xilinx router. It also generates a VHDL file, which should be used only for simulation. The number of input data bits equals the number of clock cycles required for the calculation of one output sample. Therefore, the input word size of the filters has to be 8 (a power of 2). Otherwise the timing would become much more complicated, e.g. a input wordsize of 9 bit would require a ratio of 9 between different clocks. Changing the filter coefficients requires the following actions:

1) Change the coefficients in the *.coe file, associated with each filter.
2) Generate the filter with Core Generator, copy *.edn file into the Xilinx Foundation project directory.
3) Create a new version of implementation (in the Xilinx project manager menu).

3.5 FFT/IFFT

The FFT/IFFT, which is a fast algorithm for the DFT/IDFT, was the most difficult part in the realization of the analysis/synthesis bank and there may be still possibilities for improvement. In the literature [3] there are many different methods given to implement an FFT, ranging from fully parallel (high speed, large hardware) to fully serial (slow, hardware efficient). The same tradeoff as with the filters, explained in the previous chapter, has to be made. For this project a "pipeline FFT" was selected because it provides only as much parallelism as necessary, and therefore consumes not too much area (CLBs of the FPGA). Furthermore, it is easily scaleable for different numbers of subbands. The "pipeline FFT" makes use of the fact that the calculation of the second stage of an FFT can be started as soon as the calculation of the third butterfly in the first stage is completed. This can be seen from the FFT flowgraph given in Figure 3.5.

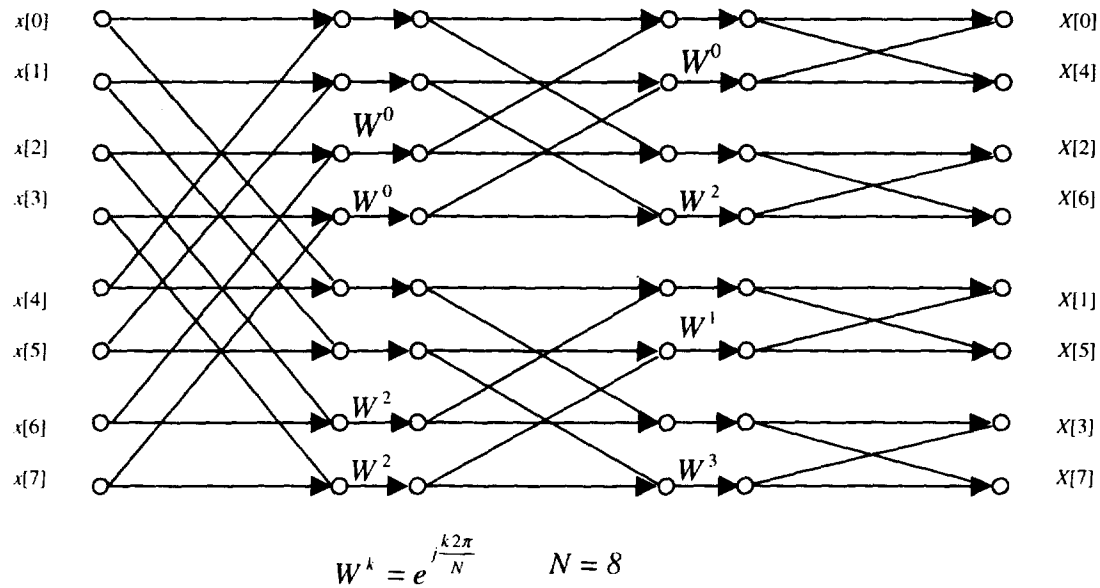


Figure 3.5: Flow graph for 8 pt. FFT

A block diagram of the "pipeline FFT" is given in Figure 3.6. Data from the analysis filters is written with a rate of clk/8 into the first (parallel in, serial out) shift registers. The output of the shift registers (serial) is moved into the pipeline with a rate of clk (the system clock). From the flow graph above it can be seen that after the calculation of the first two butterflies of the first stage the cross switch has to be toggled, and a multiplication of the lower butterfly output by j has to be performed. This operation is controlled by the delayed value of a control counter.

The content of this control counter is shifted through the pipeline parallel to the data. The FFT in bit-reversed order is available at the output shift register.

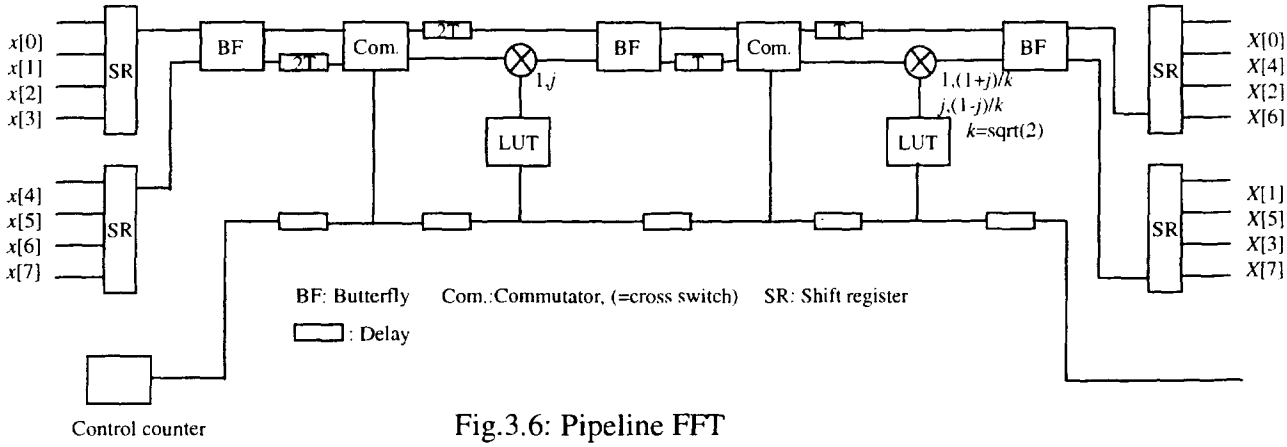


Fig.3.6: Pipeline FFT

The pipeline FFT is easily scaleable for different numbers of FFT points by adding additional pipeline stages. Figure 3.7 shows a flowchart of the VHDL code fft.vhd, Appendix B. This flowchart shows how to build hardware, not how a program executes !

It is well known that a N-point FFT of a real signal can be calculated using a $N/2$-point FFT by feeding the odd indexed values of the time sequence into the imaginary inputs of the FFT. After this $N/2$-point FFT, the FFT of the even indexed sequence and the FFT of the odd indexed sequence have to be combined to the desired FFT of the complete input sequence, which requires a complex multiplication of the odd indexed sequence with a twiddle factor. Because in a hardware implementation the number of multipliers is most important, the use of a $N/2$-point FFT has no advantage over the use of a N-point pipeline FFT, and the implementation would be more complicated. Therefore, for M subbands a M-point pipeline FFT is used.

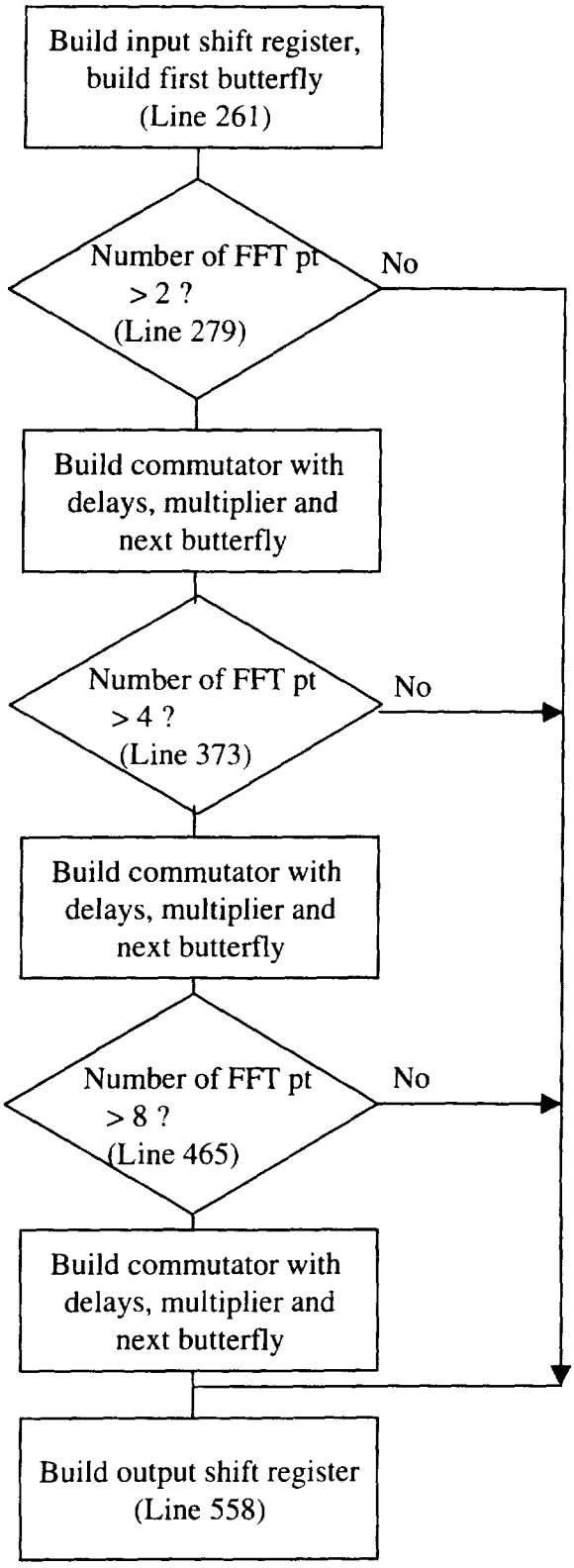

Build input shift register, build first butterfly (Line 261)
Number of FFT pt > 2 ? (Line 279)
No
Build commutator with delays, multiplier and next butterfly
Number of FFT pt > 4 ? (Line 373)
No
Build commutator with delays, multiplier and next butterfly
Number of FFT pt > 8 ? (Line 465)
No
Build commutator with delays, multiplier and next butterfly
Build output shift register (Line 558)

Figure 3.7: Flowchart of fft.vhd

3.6 Software tools used

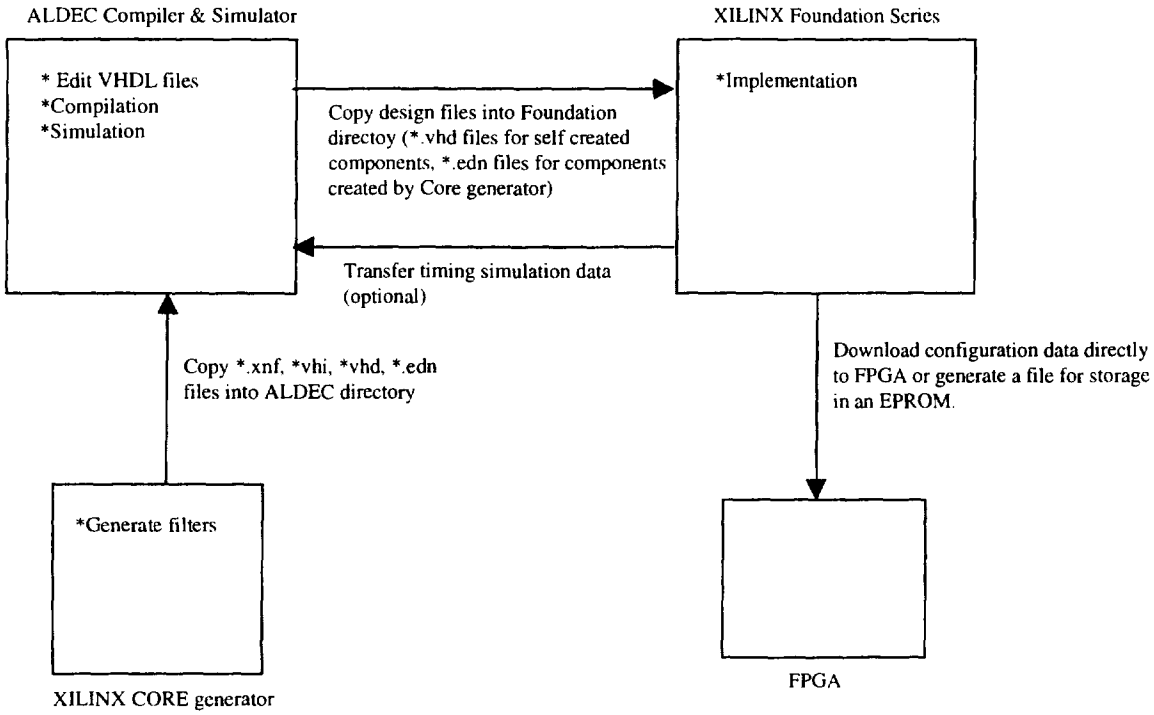

ALDEC Compiler & Simulator
* Edit VHDL files
*Compilation
*Simulation
XILINX Foundation Series
*Implementation
Copy design files into Foundation directoy (*.vhd files for self created components, *.edn files for components created by Core generator)
Transfer timing simulation data (optional)
Copy *.xnf, *vhi, *vhd, *.edn files into ALDEC directory
Download configuration data directly to FPGA or generate a file for storage in an EPROM.
*Generate filters
XILINX CORE generator
FPGA

Figure 3.8: Design tools

Figure 3.8 shows the design tools used for the project. Editing of all VHDL code, compilation and simulation is done with the Aldec compiler and simulator. The filters and the multipliers for the FFT/IFFT are generated with Core generator, a program from Xilinx. Core generator produces a VHDL file and a *.edn file. The VHDL file is only for simulation and must not be used for implementation, the *.edn file is used for implementation by the Xilinx implementation tools. After editing and simulating all VHDL code with Aldec, Xilinx Foundation Series is used to implement the circuit into the FPGA. To do this, a new project is created with the Xilinx Foundation project manager, then all files are copied from the Aldec directory to the newly created project directory, then everything is compiled again and finally implemented into the FPGA with Xilinx Foundation. The output of the Xilinx program is a bit file, which has to be downloaded to the FPGA. There are two ways to download the configuration to the FPGA. One way is to use the pins DIN, CCLK, PROG and use the Hardware debugger in the Xilinx project manager. DIN and CCLK perform a synchronous serial data transmission from the PC to the FPGA, PROG is reset line. Simply connect the pins Vcc, GND, DIN, CCLK, PROG of the download cable with the respective pins of the FPGA, then select "download config"

in the hardware debugger. The configuration is downloaded, but nothing is verified by readback.

Another way is to use the JTAG port of the FPGA and the JTAG programmer in the Xilinx project manager. This method allows readback of data from the FPGA. Communication over the JTAG port did not work, so only the first way was used here in this project.

It is also possible to store the configuration in an EPROM. After power on, the FPGA reads the data from the EPROM and configures itself. In Appendix G is a schematic, which shows how to connect the EPROM to the FPGA and how to set the jumpers on the prototype board. The EPROM data is obtained from the file *.bit, which is generated by the Xilinx implementation (XACT) tools. The content of this file from address 45h until EOF is burned into the EPROM starting at address 0. The begin of the truncated file (starting at 45h) looks like in table 20, page 4-49 of the XILINX data book [7]. The FPGA must read the EPROM data in bit-reversed order, which can be accomplished by connecting the EPROM in bit-reversed order to the FPGA, or by flipping the bytes in the file before burning the content into the EPROM. In this project, the EPROM is connected in bit-reversed order to the FPGA.

3.7 CLB counts

Table 3.2 shows the number of CLBs used for different number of subbands. The 4 subband analyzer/synthesizer was implemented using two XC 4025E devices, which contain 1024 CLBs each.

Table 3.2: Chip area for different filterbank sizes

	CLBs used		
Nmb. Of subbands	4	8	16
Analyzer without filters	273 CLB	688 CLB	1525 CLB
Filters for analyzer (32 tap)	428 CLB	856 CLB	1712 CLB
Total analyzer with filters	701 CLB	1544 CLB	3237 CLB
Synthesizer without filters	208 CLB	463 CLB	913 CLB
Filters for synthesizer (32 tap)	428 CLB	856 CLB	1712 CLB
Total synthesizer with filters	680 CLB	1407 CLB	2861 CLB

4 Testing the 4 subband analysis/synthesis filter banks

This chapter describes the testing of the analysis and synthesis filter banks with the subband signals connected by cables. This configuration was a temporary state during the development of the hardware. In the final version of the hardware, the subband signals are passed through Motorola EVMs (see chapter 5).

4.1 Sinusoidal Input

Figure 4.1 shows a block diagram of the test circuit, detailed schematics are given in Appendix G.

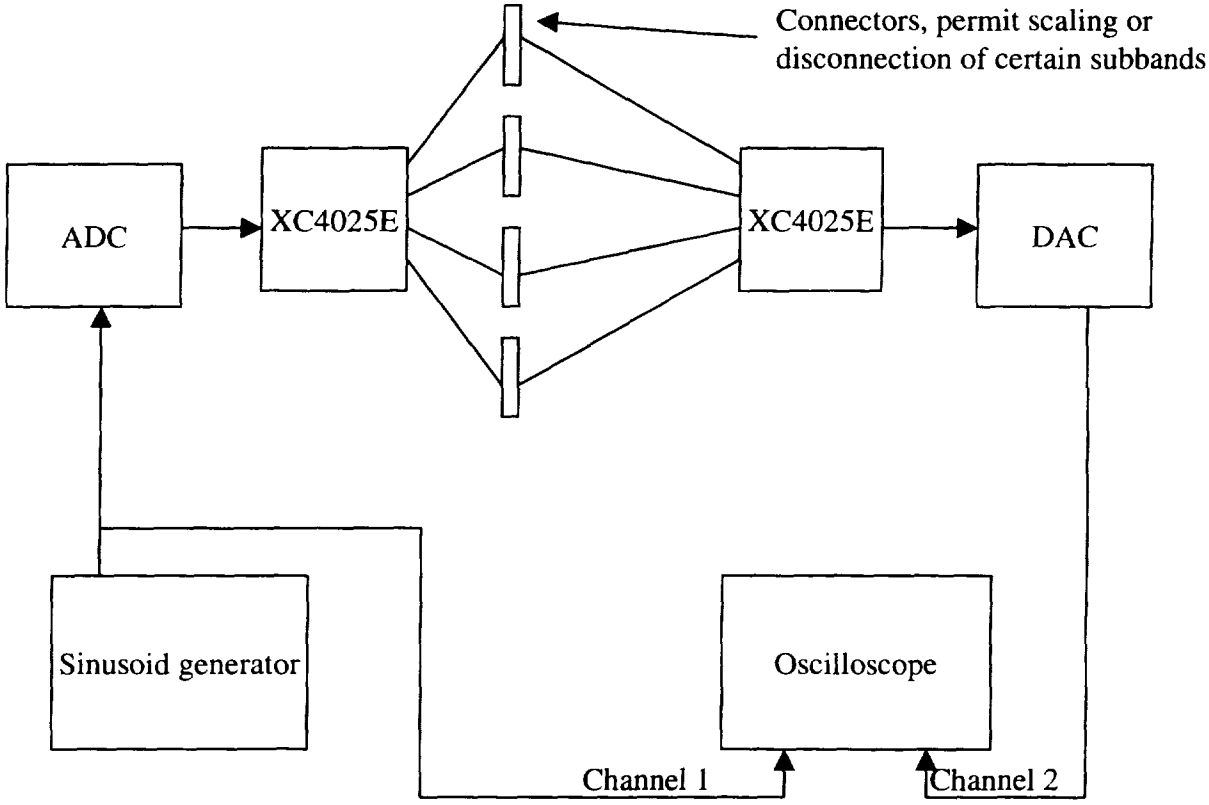


Figure 4.1: Circuit to test analysis and synthesis filter bank

The sinusoid from the sinus generator is sampled at 8 MHz, digitized with 8 bits by Analog Devices AD9048 and fed into the analysis filter bank. The subband signals, 4 MHz data rate, wordsize of the analysis bank output 16 bit, wordsize of the synthesis bank input 8 bits, are connected by cables. The connectors allow it to scale the amplitude of certain subbands by factors of 2, or disconnect certain subbands easily. An 8 bit DAC, Analog Devices ADV 101 converts the digital output of the synthesis filter bank back to an analog signal. Input and output signals are displayed with the oscilloscope. The result is a sinusoid with constant amplitude at the output for all frequencies except for a slight discontinuity near the boundary of two subbands due to crossover distortion.

Furthermore, manipulations in a particular subband does not affect any frequencies outside the subband. By scaling the amplitude of the subbands using the connectors, a magnitude response as shown in Figure 4.2 can be generated.

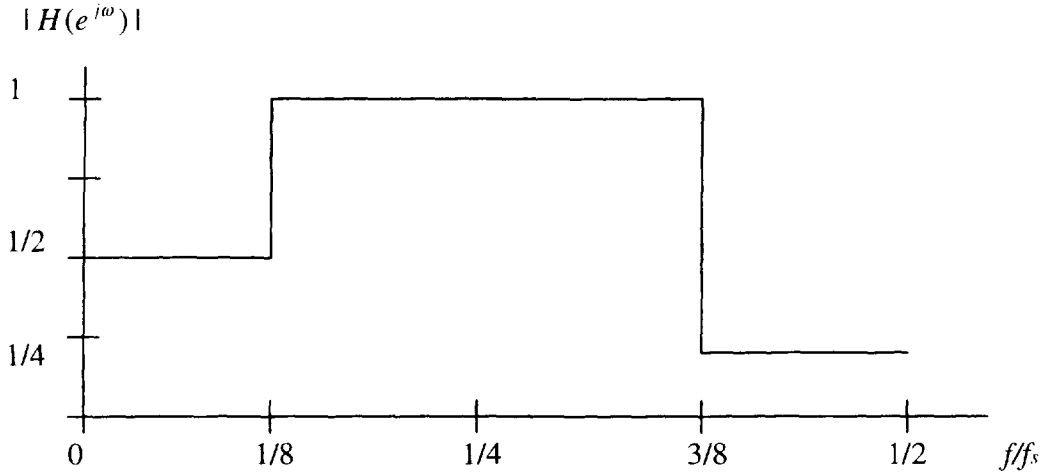


Figure 4.2: Example for a frequency response generated by scaling the subbands

Schematics of the ADC and DAC circuit are given in Appendix G. The ADC accepts a voltage from 0 to -2 V at the analog input, where 0 V leads to a digital output (2's complement) of 1000000, -1 V leads to 00000000, -2 V gives an output of 0111111. For more details about the ADC, see Analog Devices data book, vol.2, 1992, page 2-773, [7]. The input of the ADC is hold at - 1 V by the voltage divider, the signal is AC coupled, because the DC offset of the sinus generator used (Wavetek Mod. 185) is not very stable and hard to adjust to zero. All diodes in the schematic are only for protection of the ADC against overvoltage, no other function.

The DAC is an inexpensive video DAC, but it works well for other applications too. The chip contains 3 DACs for the 3 colors, 1 DAC is used, the remaining 2 DACs have their inputs grounded. Additional information can be found in the Analog Devices data book, vol.1, page 2-793, [7]. One problem interfacing this DAC was the fact that the FPGA output is in 2's complement representation, but the DAC expects unsigned binary numbers. This problem was solved by inverting the MSB before it enters the DAC. After writing down the binary numbers in 2's complement, e.g. for 4 bits, it can be seen, that after inverting the MSB, a meaningful input (for the DAC) results. This inverting of the MSB is equivalent to adding an offset to the bipolar number.

The hardware in Figure 4.1 was an intermediate state, the final version has Motorola DSP 56302 EVMs in the subbands. Therefore, exactly the same experiment with cables connecting the subbands cannot be repeated without removing the wiring on the backside of the prototype board. But the same result can be obtained with the EVMs. Shutting down one EVM while the others are working has the same effect as interrupting one subband by opening the connection. A scaling can easily programmed into the EVMs. More information about interfacing the EVMs to the FPGAs is given in Chapter 5.

The whole parallel DSP hardware can be bypassed by changing connectors near the ADC and DAC, which may be useful in order to verify the correct operation of the ADC and DAC. That means, the input signal is digitized, and the immediately converted back to an analog signal.

4.2 Impulse response

The circuit in Figure 4.1 is slightly modified to watch the impulse response of the system. Figure 4.3 shows the modified circuit.

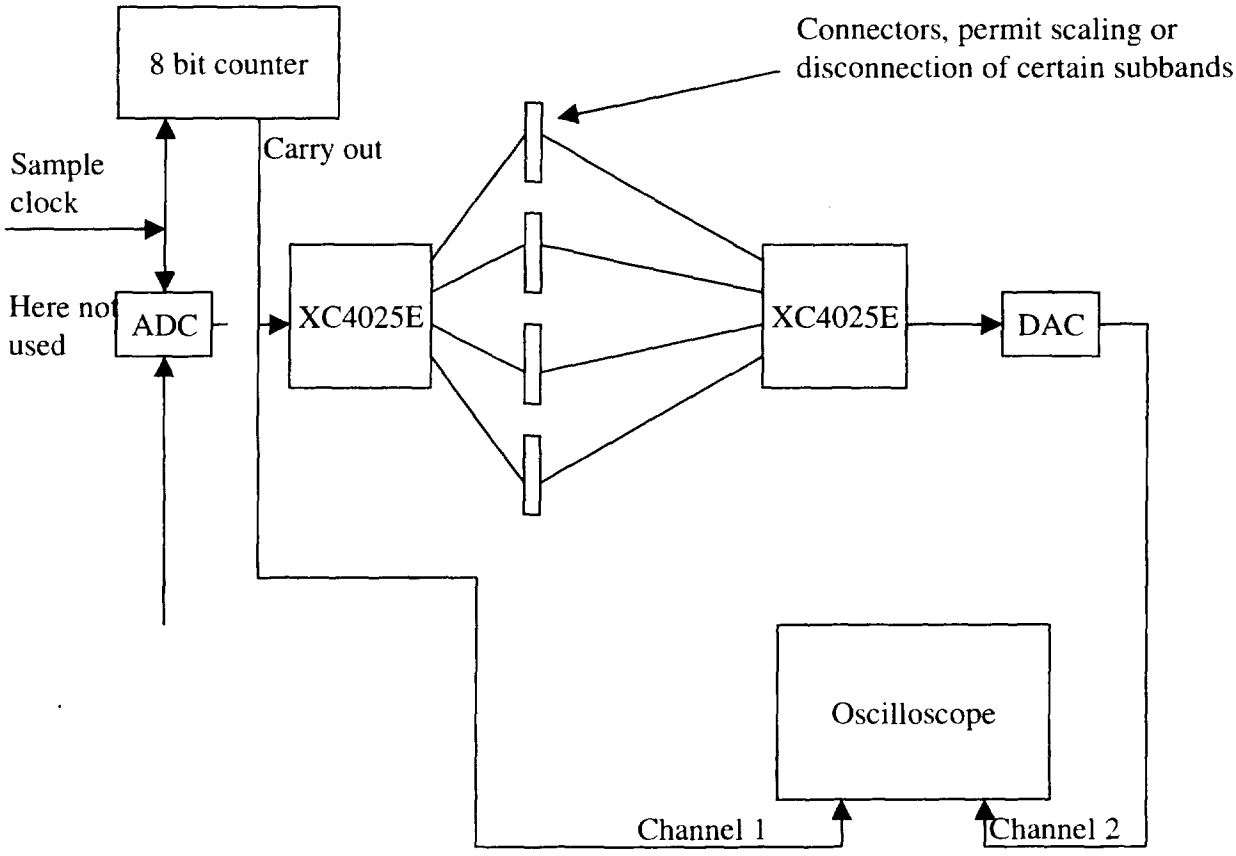


Figure 4.3: Circuit to view the impulse response

Instead of using the ADC, the carry signal of an 8 bit counter, which is clocked with the sample clock, is used to generate an input impulse train. The space between pulses (255 zeros) is big enough to allow the response to decay completely before the next pulse is applied, the space is small enough to display the response with every analog oscilloscope. The impulse response was captured with a digital scope and is given in Figure 4.4. It can be seen, that the impulse response of the hardware matches with the Matlab simulation in Appendix C and the Aldec simulation in Appendix D. The numbers in the Aldec simulation plot are in 2's complement representation. The input pulse in the Matlab simulation is normalized to 8 bit ($7F = 0.9921875 \approx 1$ in fractional interpretation, 7F = 127 in integer interpretation).

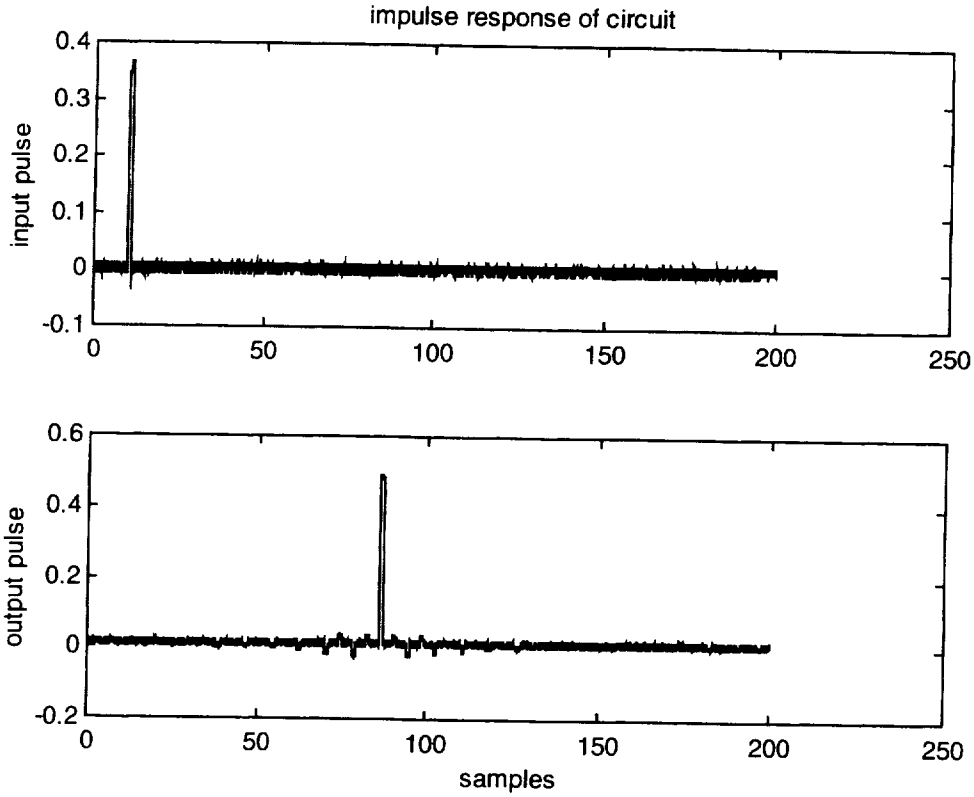


Figure 4.4: Impulse response of hardware

5 Using DSPs for subband processing

5.1 Interfacing DSPs to the FPGAs

For more sophisticated subband processing, a DSP for each subband may be desirable. Figure 5.1 shows how to interface a Motorola DSP 56302 EVM to the FPGAs. The analysis filter bank writes data into an output register, the falling edge of the register clock causes a hardware interrupt of the DSP, the DSP reads the register content, performs processing and writes the processed subband data to an input register of the synthesis filter bank.. A pass through data rate of 4 MHz, that means the DSP performs only data transfer, has been achieved. This implies a fullband rate of 8 MHz. All logic required for interfacing is integrated into the FPGAs (see Fig. 5.1 and the VHDL files an_n.vhd, sy_n.vhd, Appendix B).

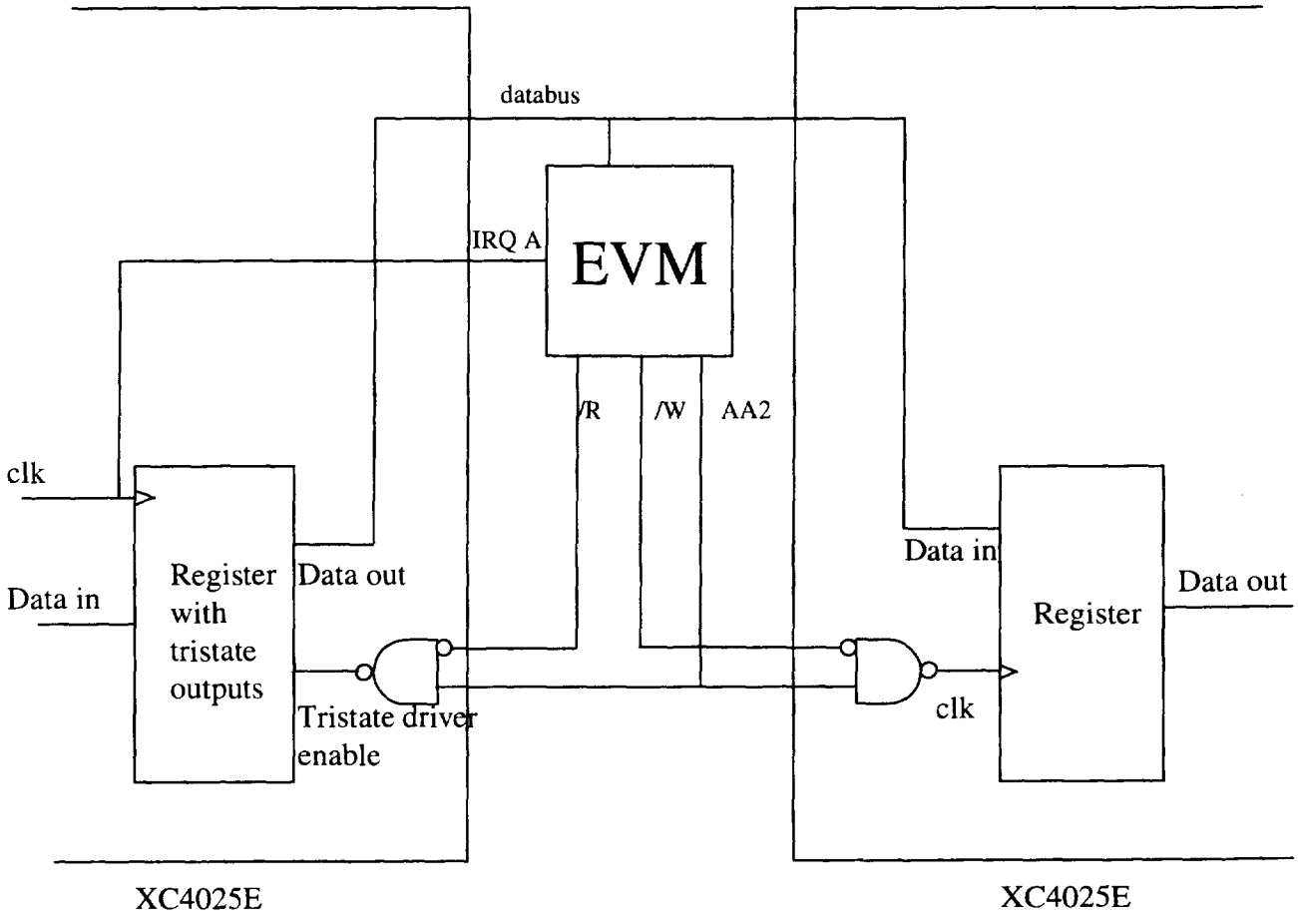


Figure 5.1: Interfacing DSP Evaluation board to the FPGAs

In Appendix F, a block diagram of the hardware is given. Only 3 EVMs are needed due to the Hermitian symmetry of the spectrum (real input signal). See Figure 1.2. The 2nd subband (filter $\mathbf{f}_1$) is related to the 4th subband (filter $\mathbf{f}_3$), e.g. the real part of the 4th subband is the same as the real part of the 2nd subband, the imaginary part of the 4th subband is the negative of the imaginary part of the 2nd subband. The 4th subband signal of the analysis bank is ignored. In the synthesis FPGA, the 4th subband signal is recovered from the 2nd subband signal. See Appendix B, Sy_n.vhd, line 204.

A schematic with detailed connections and instructions about jumper settings is given in Appendix G. Notice the difference between subbands with real and subbands with complex signals, given in the Appendix.

It is important to remember, that the IRQA pin is a mode select pin during reset and must be kept high, if the EVM is supposed to execute the code, which is in the EEPROM of the EVM. The IRQA pins of the EVMs are connected to the An_rdy_out signal, which is kept high during reset of the FPGAs (see Appendix B, An_n.vhd, line185). Therefore, starting up the system goes in the following order. Hold the FPGAs in the reset state by shortening the reset pin to GND, reset the EVMs by cycling the power, start the FPGAs by releasing the reset pin. During loading code into the EVMs and writing the EEPROM, hardware interrupts to the EVMs should be suppressed, otherwise trouble may occur. This is also accomplished by holding the FPGAs in the reset state.

Another problem is the signal quality on the data bus during read and write from/to the FPGAs. Without wait states, the /RD and /WR pulses have a duration of about 15 ns (1/66 Mhz). This is a very short time to get valid data to the EVM (for read) or to the FPGA (for write). One problem is the inductance of the wires (wire wrap board, long wire runs), a well designed PCB is a necessary condition. Another issue is the slew rate of the drivers, must select "high slew rate drivers" in the *.ucf file (user constraint file) from Xilinx project manager. Biggest problem are the (misdesigned) EVM PCBs without any GND pin. Normally, every signal should have its own GND pin for such high speed data transmission. Only solution is to search several GND points on the EVM PCB and connect them to the GND point of the other hardware. The hardware built here does not work without inserting several wait states (see Appendix E, sub_pass.asm, line 185).

Another big problem of interfacing the Motorola 56k EVMs (3 Volt logic) with the XC4025E (5 Volt logic) was to protect the inputs of the EVM from damage by over voltage. Here, the operating voltage of the XC4025E devices was reduced to 3 Volt in order to allow interfacing without any additional level conversion. This is a very unusual method (operating a 5 Volt device with 3 Volt), of course, the performance of the FPGAs is reduced. But the goal here was to test the interface method. Later, in a future version of this hardware, 3 Volt FPGAs will be used and interfacing will be no problem.

Configuring the FPGAs using EPROMs and 3 V supply requires some "trick".
There is a comparator inside the FPGA, which starts configuration as soon as the supply voltage exceeds 3.5 V, starting up the system has to be done in the following way.

1) Switch power on, adjust the voltage to 4 V (it is assumed that this does not damage the EVMs) .
2) Cycle the power. The FPGA loads the configuration data from the EPROM.
3) Reduce the voltage to 3 V.

The Motorola 56k code for pass through of the subband signals, sub_pass.asm, is given in Appendix E. This program is a modified version of pass.asm. Until line 135 nothing is changed. The part after line 135 is new, mostly initialization of the hardware interrupt and address attribute pin #2 (AA2). Included in sub_pass.asm is the routine flash.asm, which writes the code to the EEPROM of the EVM. See the comments in the file sub_pass.asm how to accomplish this. There are 3 options to handle the hardware interrupt (IRQA):

1) "Fast interrupt"
   The code to handle the interrupt is written directly into the vector table, maximum code size are 2 words. Only useful for trivial signal processing, like pass through.
2) Jump to Interrupt Service Routine (ISR)
   The vector table contains a jump instruction to an ISR. All signal processing is performed in the ISR.
3) Set flag to indicate new sample
   The vector table contains an instruction, which sets a "new sample flag". This flag is frequently checked in the main loop of the program. If it is set, processing is performed, and then the flag is cleared.

Except for trivial signal processing, method 2 or 3 has to be applied. The author experienced problems using method 2, e.g. certain instructions executed in an ISR lead to a system crash. For example, any access to register a in an ISR crashed the system according to the author's experience, reason not clear for the author. Therefore, the author prefers method 3.
See also the comments in the code.

5.2 Operating the complete HW

This section explains the operation of the complete HW (analysis filter bank, Motorola EVMs for subband processing, synthesis filter bank), which is a little bit complicated due to the 3 V operation of the 5 V FPGAs, and due to the second function of the IRQA pin (mode select during reset).

Powering up:

1) Connect power supplies:
   Red cable (TTL ICs, ADC): + 5 V.
   Orange cable (FPGA supply): + 3 V.
   Brown cable (ADC): -5 V.
   EVMs: The usual 9 V supply.

2) Switch on in following order: EVM supply, + 3 V supply, +/- 5 V supply.

3) Increase the +3 V supply to + 4 V, cycle this power, reduce this supply to + 3 V. Necessary, if loading the configuration from the EPROM is desired.

EVM reset:

1) Bring the FPGAs into reset state by shortening the reset pin of the FPGAs to GND (with jumper). This ensures that the IRQA pin is high, and the EVMs load the program from their EEPROM after reset.

2) Reset the EVMs by cycling the power.

3) Release the reset pin of the FPGAs.

Remark:
The 2nd EVM (the one with the damaged CODEC) loses the content of the EEPROM after a relative short time (some weeks) according to the author's experience. It may be necessary to reload the program. This is done in the following way:

1) Bring FPGAs into reset state

2) Connect EVM to serial port of PC, start debugger, load sub_pass.cld into the EVM

3) Set the program counter (PC) to 800h. This is the address, where the code for loading the EEPROM (from flash.asm) begins.

4) Select "run".

5) Select "stop"

The debugger should display that the EVM was executing the instruction "bra *", which is preceded by the instruction "debug" (see line 87 in flash.asm, Appendix E). Otherwise, the EEPROM is not correctly programmed.

6) Reset the EVM by cycling the power.

Sinusoidal Input:

1) Connect the output of the sinusoid generator to the input of the circuit (to the ADC), to channel A of the oscilloscope and to the frequency counter. The sinusoid should have a level of about 1 V pp.

2) Connect the output of the circuit (DAC) to channel B of the oscilloscope.

3) The sinusoid displayed on channel B may appear noisy, including spikes. Make a fine adjustment of the + 3 V supply (FPGA supply) in order to minimize the amount of spikes. Try also to shift the downsampler in the synthesis bank by 1 sample using the jumper. This may be necessary due to the fact that the reset signal of the FPGAs is not synchronized with the sample clock. A synchronization could be done easily with a D FlipFlop, clocked with the sample clock.

4) Sweep the sinusoid generator over the frequency band of interest, verify that all EVMs are working. If not, reset the respective EVM again.

Shutdown of particular EVM:

Cycle the power of the EVM, while the FPGA is operating. The IRQA pin, which is mode select pin during reset, is low most of the time, when the FPGAs are working. Therefore, the EVM does not load the program from its EEPROM and does not work.

Impulse response:

1) Remove sinusoid from circuit input and from channel A of the oscilloscope.

2) Remove the connector on the prototype board, which connects the ADC to the analysis FPGA.

3) Feed the carry signal of the counter into the analysis FPGA.

4) Display the carry out signal also on channel A of the oscilloscope.

6 Conclusion

6.1 Summary

This work presents a hardware for parallel digital signal processing, built with relatively inexpensive "off the shelf" components. An analysis filter bank and a synthesis filter bank are implemented with Xilinx FPGAs, subband processing is performed with Motorola DSP 56302 EVMs, which are interfaced to the FPGAs without any additional logic. This hardware has been built and verified against Matlab simulations for 4 subbands. An extension to higher number of subbands, e.g. 8 or 16, is done easily due to a scaleable design using VHDL.

6.2 Future work

It turned out that the calculation of the DFT/IDFT using the FFT/IFFT requires a major part of the available hardware resources (CLBs), especially for higher number of subbands. One area of future work may be to improve the efficiency of the DFT/IDFT calculation. It seems unlikely to achieve much improvement of the FFT/IFFT calculation, considering the large number of researchers, who tackled this problem in the last 30 years. A more recently published paper [9],[11] presents a completely different way to calculate the DFT/IDFT, which may be useful here.

Another area of improvement is the use of other filter coefficients, obtained with more sophisticated filter design methods, in order to achieve a better suppression of the sidelobes in the impulse response.

Finally, this hardware should be tested by implementing some useful application with it. An example for an application can be found in [12].

7 References

[1] Phillip L. De Leon, "On the use of Filter Banks for Parallel Digital Signal processing", 7th NASA Symposium on VLSI Design 1998.

[2] L. Rabiner and R.Crochiere, "Multirate digital signal processing", Englewood Cliffs, NJ, Prentice-Hall 1983.

[3] Bede Liu, "Digital filters and the fast Fourier transform", Dowden, Hutchinson & Ross, Inc., Stroudsburg, Pennsylvania, 1975.

[4] Motorola, "DSP 56300 Family Manual", Motorola Semiconductor Products Sector, 1995.

[5] Motorola , "DSP 56302 EVM, User's Manual, Rev. 1", Motorola Semiconductor Products Sector.

[6] P. P. Vaidyanathan, "Multirate Systems and Filter Banks", Englewood Cliffs, NJ, Prentice Hall 1993.

[7] Xilinx, "The programmable logic data book 1998", Xilinx, San Jose, CA, 1998

[8] Analog Devices, "1992 data Converter Reference Manual", Vol.1-2, Analog Devices, Norwood, MA, 1992.

[9] Andreas Dandalis and Viktor K.Prasanna, "Fast Parallel Implementation of DFT Using Configurable Devices", 7th International Workshop, FPL '97, London, UK, 1997.

[10] Gregory Ray Goslin, "A Guide to Using Field Programmable Gate Arrays (FPGAs) for Application-Specific Digital Signal Processing Performance", Xilinx, San Jose, CA, 1995.

[11] I. S. Reed, D. W. Tufts, X. Yu, T. K. Truong, M. Shih and X. Yin, "Fourier analysis and signal processing by use of the Moebius inversion formula", IEEE Trans. Acoust., Signal Processing, Vol.38, no. 3, pp. 458-470, Mar. 1990.

[12] R.Sadr, P. P. Vaidyanathan, D. Raphaeli, S. Hinedi, "Parallel digital modem using multirate digital filter banks", JPL Publication 94-20, Aug. 1994.

Appendix A
MATLAB code to calculate the polyphase filter coefficients and plot the impulse response

```
function [an_coeff,sy_coeff] = ..
  fb(nmb_of_subb,oversampl_factor,filt_proto_length,coeff_size,scale,delay)
%  This program calculates the coefficients of the polyphase filters for
%  an analysis filter bank and an synthesis filter bank from a given
%  lowpass prototype. It also plots the impulse response of the whole
%  system (analysis bank + synthesis bank). The program considers the
%  finite length of the filter coefficients and the rounding of the
%  subband signals, but it does not consider the finite precision inside
%  the filters and the FFT/IFFT. To do this, the functions polyph, ppan,
%  ppsynthe, written by Dr. DeLeon, have to be modified.
%
%  Input parameters:  nmb_of_subb:          Number of subbands,
%                                           e.g. 2,4,8,16,...
%                     oversampl_factor:     Oversample factor,
%                                           e.g. 1,2,4,...
%                     filt_proto_length:    Length of the used lowpass
%                                           prototype, should be integer
%                                           multiple of the number of
%                                           subbands, other lengths lead to
%                                           "ugly" impulse responses, which can
%                                           be verified by experiment.
%                     coeff_size:           number of bits used for the filter
%                                           coefficients.
%                     scale:                number of bits cut of before the
%                                           subband signals enter the
%                                           synthesis bank
%                     delay:                delays the input pulse by "delay"
%                                           samples, neccessary, because the
%                                           combination of the analysis and
%                                           synthesis bank is time variant
%
%  Output parameters: an_coeff:             coefficients of the polyphase
%                                           filters of the analysis bank
%                     sy_coeff:             coefficients of the polyphase
%                                           filters of the synthesis bank
%
%  For the 4 subband, 2 times oversampled analysis and synthesis filter banks,
%  which have been built, call:
%  [an_coeff,sy_coeff]=fb(4,2,64,8,16,0)


   f = fir1(filt_proto_length,1/nmb_of_subb)'          %generate lowpass
                                                       %prototype
%   f = f.*hamming(length(f));
   f_norm = round(f*(2^(coeff_size-1)-1)/max(abs(f))); %consider finite
                                                       %coefficient
                                                       %quantization


   x = [zeros(delay,1); 127; zeros(511-delay,1)];      %input impulse

  [an_coeff,sy_coeff] = ..
      polyph(f_norm,f_norm,nmb_of_subb,nmb_of_subb/oversampl_factor);
   xx = ppan(an_coeff,oversampl_factor,x);

   xx = round(xx)/2^scale;                             %round the subband
                                                       %signals
```

```
58
59      y = ppsynthe(sy_coeff,oversampl_factor,xx);
60      y = round(y);
61
62      subplot(2,1,1);
63      %stem(20*log10(abs(x)));
64      stem(x);
65      ylabel('input pulse');
66      title('impulse response');
67
68      subplot(2,1,2);
69      %stem(20*log10(abs(y)));
70      stem(y);
71      ylabel('output pulse');
72      xlabel('samples');
73
74
```

```
function [a,r] = polyph(f,g,M,D)   %renamed
%************************************************************************
% POLYPHASE [a,r] = polyphase(f,g,M,D)
%
% Description: This function computes the M polyphase analysis filters a,
synthesis filters r from the
%  given prototype, f, g respectively assuming a downsampling factor of D and
clockwise
%  comutator, counter-clockwise commutator respectively.  M/D must be an
integer.
%
% Call Syntax: [a,r] = polyphase(f,g,M,D)
%
% Input Arguments:
%     Name: f
%     Type: matrix
%     Description: Prototype analysis filter stored as column vector.
%
%     Name: g
%     Type: matrix
%     Description: Prototype synthesis filter stored as column vector.
%
%     Name: M
%     Type: scalar
%     Description: Number of subbands (integer)
%
%     Name: D
%     Type: scalar
%     Description: Downsampling factor (integer).
%
% Output Arguments:
%     Name: a
%     Type: matrix
%     Description: Polyphase (clockwise) analysis filters stored as column
vectors.
%
%     Name: r
%     Type: matrix
%     Description: Polyphase (counterclockwise) synthesis filters stored as
column vectors.
%
% Programmer: Phillip De Leon
%
% Creation Date: June 22, 1998
% Last Revision:
%
% Required subroutines:
%
% Notes: f, g and a, r can be flipped so that analysis filter employs a
counterclockwise
%   commutator and synthesis filter employs a clockwise commutator.
%
% References: Rabiner and Crochiere p.311 (7.59)
%
% Copyright (c) 1998 by Phillip De Leon, All Rights Reserved
%************************************************************************
```

```
%------------------
% Check valid input
%------------------
if (nargin ~= 4)
   error('Error (polyphase): must have four arguments: prototype analysis and
synthesis filters, number of subbands, and downsampling factor.');
end;

%-----------
% Initialize
%-----------
[L1,MM]= size(f);
if (MM ~= 1)
   error('Error (polyphase): prototype analysis filter should be a column
vector.');
end;
[L2,MM]= size(g);
if (MM ~= 1)
   error('Error (polyphase): prototype synthesis filter should be a column
vector.');
end;
if (M/D ~= ceil(M/D))
   error('Error (polyphase): M/D must be an integer.');
end;

%--------------------------------------
% Generatate Clockwise Polyphase Filters
%--------------------------------------
a = zeros(floor(L1/D)+D,M);
for m = 0:M-1
   polyphase_indices = [-m+1:D:L1];
   P = length(polyphase_indices);
   i = length(find(polyphase_indices < 1));  % # indices < 1
   a(:,m+1) = [zeros(i,1);f(polyphase_indices(i+1:P));zeros(floor(L1/D)+D-
P,1)];
end;

%----------------------------------------------
% Generatate Counter-Clockwise Polyphase Filters
%----------------------------------------------
r = zeros(floor(L2/D)+D-1,M);
for m = 0:M-1
   polyphase_indices = [m+1:D:L2];
   P = length(polyphase_indices);
   r(:,m+1) = [g(polyphase_indices);zeros(floor(L2/D)+D-P-1,1)];
end;
```

```
function xx = ppan(a,I,x)   %renamed
%**************************************************************************
% ANALYZE_P xx = analyze_p(a,I,x)
%
% Description: This function analyzes the signal, x by
%  filtering with polyphase filters in a with an oversampling factor of I
%  using a counterclockwise commutator. The usual use for this routine is
%  for analysis filtering.
%
% Call Syntax: xx = analyze_p(a,I,x)
%
% Input Arguments:
%     Name: a
%     Type: matrix
%     Description: Set of analysis polyphase filters stored as column vectors.
%
%     Name: I
%     Type: scalar
%     Description: Oversampling factor (integer)
%
%     Name: x
%     Type: matrix
%     Description: Input signal stored as column vector.
%
% Output Arguments:
%     Name: xx
%     Type: matrix
%     Description: Subband signals stored as column vectors.
%
% Programmer: Phillip De Leon
%
% Creation Date: June 22, 1998
% Last Revision:
%
% Required subroutines:
%
% Notes:  To compute the impulse response of a filter bank use:
%   [a,r] = polyphase(f(:,1),g(:,1),M,D);xx = analyze_p(a,M/D,[1
zeros(1,1023)]');y = D*synthesize_p(r,M/D,xx);
%
%   This implementation of the polyphase, uniform-DFT filter bank
%   assumes a clockwise commutator.
%
% References: Rabiner and Crochiere
%
% Copyright (c) 1998 by Phillip De Leon, All Rights Reserved
%**************************************************************************

%------------------
% Check valid input
%------------------
if (nargin ~= 3)
   error('Error (analyze): must have three arguments: analysis polyphase
filters, oversampling factor, and input signal.');
end;

%-----------
```

```
% Initialize
%-----------
signal_len = length(x);
[N,M] = size(a);
ssignal_len = ceil((signal_len+M-1)/M)*I;
xx = zeros(ssignal_len,M);

%---------------------------------
% Polyphase Uniform-DFT Filter Bank
%---------------------------------
% Get polyphase filter outputs
for m = 1:M
   tmp = downsa([zeros(M-m,1);x;zeros(m-1,1)],M); % delay before downsample
   %tmp = downsa([zeros(m-1,1);x;zeros(M-m,1)],M); % delay before downsample
   tmp2 = upsample(tmp,I);
   xx(:,m) = filter(a(:,m),1,tmp2);
end;

% DFT processing
for n = 1:ssignal_len
   xx(n,:) = fft(xx(n,:));
end;
```

```
function y = ppsynthe(r,I,xx)    %renamed
%**************************************************************************
% SYNTHESIZE_P y = synthesize_p(r,I,xx)
%
% Description: This function synthesizes the subband signals, xx by
%  filtering with polyphase filters in a with an oversampling factor of I.
%  The usual use for this routine is for synthesis filtering.
%
% Call Syntax: y = synthesize_p(r,I,xx)
%
% Input Arguments:
%     Name: r
%     Type: matrix
%     Description: Set of synthesis polyphase filters stored as column vectors.
%
%     Name: I
%     Type: scalar
%     Description: Oversampling factor (integer)
%
%     Name: xx
%     Type: matrix
%     Description: Subband input signals stored as column vectors.
%
% Output Arguments:
%     Name: y
%     Type: matrix
%     Description: Output signal stored as column vector.
%
% Programmer: Phillip De Leon
%
% Creation Date: June 22, 1998
% Last Revision:
%
% Required subroutines:
%
% Notes:  To compute the impulse response of a filter bank use:
%   [a,r] = polyphase(f(:,1),M,D);xx = analyze_p(a,M/D,[1 zeros(1,1023)]');y =
synthesize_p(r,M/D,xx);
%
%   This implementation of the polyphase, uniform-DFT filter bank
%   assumes a clockwise commutator.
%
% References: Rabiner and Crochiere
%
% Copyright (c) 1998 by Phillip De Leon, All Rights Reserved
%**************************************************************************

%------------------
% Check valid input
%------------------
if (nargin ~= 3)
   error('Error (synthesize): must have three arguments: synthesis polyphase
filters, oversampling factor, and subband input signals.');
end;

[signal_len,M] = size(xx);
[N,M1] = size(r);
```

```
58  if (M ~= M1)
59     error('Error (synthesize): number of polyphase filters must equal number of
60  subband signals.');
61  end;
62
63  %-----------
64  % Initialize
65  %-----------
66  ssignal_len = signal_len*I;
67  x = zeros(ssignal_len,1);
68  ifft_out = zeros(signal_len,M);
69
70  %----------------------------------
71  % Polyphase Uniform-DFT Filter Bank
72  %----------------------------------
73  % IDFT processing
74  for n = 1:signal_len
75     ifft_out(n,:) = ifft(xx(n,:));
76  end;
77
78  % Get polyphase filter outputs
79  polyphase_filter_output = zeros(signal_len,M);  % malloc
80  yy = zeros(signal_len/I*M+M-1,M);  % malloc
81  for m = 1:M
82     polyphase_filter_out(:,m) = filter(r(:,m),1,ifft_out(:,m));
83     tmp = downsa(polyphase_filter_out(:,m),I);
84
85     yy(:,m) = [zeros(m-1,1);upsample(tmp,M);zeros(M-m,1)];  % stagger outputs,
86  i.e. delay chain
87     %yy(:,m) = [zeros(M-m,1);upsample(tmp,M);zeros(m-1,1)];  % stagger outputs,
88  i.e. delay chain
89  end; .
90  y = M*sum(yy')';  % IFFT scaling factor
```

Appendix B
VHDL code

```
1   -------------------------fltb_par.vhd-------------------------------------
2   --This file defines only constants and data types, like header file in C
3
4   library IEEE;
5   use IEEE.STD_LOGIC_1164.all;
6   use IEEE.STD_LOGIC_arith.all;
7   use IEEE.STD_LOGIC_signed.all;
8
9
10
11  package fltb_par is
12      constant nmb_of_subband:INTEGER:=4;        --set by user
13
14      constant An_in_ws:INTEGER:=8;              --set by user
15      constant An_filt_in_ws:INTEGER:=8;         --must be 8 for all possible
16                                                 --number of subb.!!!
17      constant An_filt_out_ws:INTEGER:=21;       --output wordsize of analyzer
18                                                 --filter
19      constant An_out_ws:INTEGER:=24;            --set by user
20      --for full precision An_out_ws = An_filt_out + log2(nmb_of_subband)
21
22
23      constant Sy_in_ws:INTEGER:=8;              --set by user
24      constant Sy_filt_in_ws:INTEGER:=8;         --must be 8! Don't change !!!
25      constant Sy_filt_out_ws:INTEGER:=21;       --output wordsize of
26                                                 --synthesizer filters
27      constant Sy_out_ws:INTEGER:=22;            --set by user
28      --for full precision An_out_ws = An_filt_out + log2(nmb_of_subband)
29
30      --The filters for different size of filterbank can have different
31      --output wordsizes, e.g. the set of filters for the 4 subband
32  filterbank
33      --can have output wordsize 20 bit and the set of filters for the 8
34  subband
35      --filterbank can have output wordsize 18 bit.
36      --Example:
37      --If it is desired to build and simulate the 4 subband filterbank,
38      --set: nmb_of_subband = 4
39      --     An_filt_out_ws = 20 (required for full precision)
40      --     Sy_filt_out_ws = 20
41      --If it is desired to build and simulate the 8 subband filterbank,
42      --set: nmb_of_subband = 8
43      --     An_filt_out_ws = 18 (need to save resources, reduce output size)
44      --     Sy_filt_out_ws = 18
45      --The input wordsize of the analyzer and synthesizer filter must be
46      --always 8 bit because the timing depends on this number !!!
47
48      constant COMM_OFFSET:INTEGER:=1;  --for 4 subband filterbank
49  --    constant COMM_OFFSET:INTEGER:=7;  --for 8 subband filterbank
50
51  -------------------ANALYZER-----------------------------------------------
52  ----Declarations for Com_An-----------------------------------------------
53      constant Com_An_ws:INTEGER:=An_in_ws;
54      type Com_An_out_data is array (INTEGER range 0 to nmb_of_subband-1)
55         of std_logic_vector(Com_An_ws-1 downto 0);
56  ----for An_fil4-----------------------------------------------------------
57      type An_fil_in_data is array (INTEGER range 0 to nmb_of_subband-1)
```

```
58          of std_logic_vector(An_filt_in_ws-1 downto 0);
59        type An_fil_out_data is array (INTEGER range 0 to nmb_of_subband-1)
60          of std_logic_vector(An_filt_out_ws-1 downto 0);
61    ----Declarations for FFT-----------------------------------------------------
62        type int_array2 is array (INTEGER range 0 to 16) of INTEGER;
63        constant log:int_array2:=(0,0,1,0,2,0,0,0,3,0,0,0,0,0,0,0,4);
64
65        constant nmb_of_FFT_pt:INTEGER:=nmb_of_subband;
66        constant fft_in_ws:INTEGER:=An_out_ws; --An_filt_out_ws;
67        constant fft_out_ws:INTEGER:=fft_in_ws+log(nmb_of_FFT_pt);
68        type fft_in_data is array (INTEGER range 0 to nmb_of_FFT_pt-1)
69          of std_logic_vector(fft_in_ws-1 downto 0);
70        type fft_out_data is array (INTEGER range 0 to nmb_of_FFT_pt-1)
71          of std_logic_vector(fft_out_ws-1 downto 0);
72    --    type An_out_data is array (INTEGER range 0 to nmb_of_subband-1)
73            --of std_logic_vector(An_out_ws-1 downto 0);
74        type An_out_data is array (INTEGER range 0 to nmb_of_subband/2)
75          of std_logic_vector(An_out_ws-1 downto 0);
76    --    type Rd_not_type is array (INTEGER range 0 to nmb_of_subband-1)
77            --of std_logic;
78    --    type Adr_An_type is array (INTEGER range 0 to nmb_of_subband-1)
79            --of std_logic;
80        type Rd_not_type is array (INTEGER range 0 to nmb_of_subband/2)
81          of std_logic;
82        type Adr_An_type is array (INTEGER range 0 to nmb_of_subband/2)
83          of std_logic;
84
85
86
87
88    -------------------------SYNTHESIZER-------------------------------------
89     --   constant Sy_in_ws:INTEGER:=Sy_filt_in_ws-log(nmb_of_FFT_pt)-1;
90    --    type Wr_not_type is array (INTEGER range 0 to nmb_of_subband-1)
91            --of std_logic;
92    --    type Adr_Sy_type is array (INTEGER range 0 to nmb_of_subband-1)
93            --of std_logic;
94        type Wr_not_type is array (INTEGER range 0 to nmb_of_subband/2)
95          of std_logic;
96        type Adr_Sy_type is array (INTEGER range 0 to nmb_of_subband/2)
97          of std_logic;
98
99
100       --type clk2_type is array (INTEGER range 0 to nmb_of_subband-1)
101         --of std_logic;
102   --    type Sy_in_data is array (INTEGER range 0 to nmb_of_subband-1)
103         --of std_logic_vector(Sy_in_ws-1 downto 0);
104       type Sy_in_data is array (INTEGER range 0 to nmb_of_subband/2)
105         of std_logic_vector(Sy_in_ws-1 downto 0);
106   ----Declarations for IFFT----------------------------------------------------
107    --   constant nmb_of_IFFT_pt:INTEGER:=nmb_of_subband;
108       constant IFFT_in_ws:INTEGER:=Sy_in_ws;
109       constant IFFT_out_ws:INTEGER:=IFFT_in_ws+log(nmb_of_FFT_pt);
110       type IFFT_in_data is array (INTEGER range 0 to nmb_of_FFT_pt-1)
111         of std_logic_vector(IFFT_in_ws-1 downto 0);
112       type IFFT_out_data is array (INTEGER range 0 to nmb_of_FFT_pt-1)
113         of std_logic_vector(IFFT_out_ws-1 downto 0);
114   ----for Sy_fil4--------------------------------------------------------------
```

```
115       type Sy_fil_in_data is array (INTEGER range 0 to nmb_of_subband-1)
116          of std_logic_vector(Sy_filt_in_ws-1 downto 0);
117       type Sy_fil_out_data is array (INTEGER range 0 to nmb_of_subband-1)
118          of std_logic_vector(Sy_filt_out_ws-1 downto 0);
119    ----Declarations for dwsby2-----------------------------------------------
120       constant dwsby2_ws:INTEGER:=Sy_out_ws;  --Sy_filt_out_ws;
121       type dwsby2_data is array (INTEGER range 0 to nmb_of_subband-1)
122          of std_logic_vector(dwsby2_ws-1 downto 0);
123    ----Declarations for Com_Sy-----------------------------------------------
124       constant Com_Sy_ws:INTEGER:=dwsby2_ws;
125
126
127
128    ------------------------MISCELLANEOUS-------------------------------------
129       type int_array is array (INTEGER range 0 to 63) of INTEGER;
130
131
132    end fltb_par;
133
134    --------------------------------------------------------------------------
135
136
137
```

```
1  ------------------------------------------------------------------------
2  ------------------------fb_n.vhd----------------------------------------
3  ------------------------------------------------------------------------
4  --This file describes the combination of analysis bank and synthesis bank.
5  --Only used for simulation in the ALDEC environment, because the analysis
6  --filter bank and the synthesis filter bank are implemented in different
7  --FPGAs.
8
9  library IEEE;
10 use IEEE.STD_LOGIC_1164.all;
11 use IEEE.STD_LOGIC_arith.all;
12 use IEEE.STD_LOGIC_signed.all;
13
14 library work;
15 use work.fltb_par.all;
16
17
18 entity fb_n is
19   port(
20        clk :in STD_LOGIC;
21        reset:in STD_LOGIC;
22        fb_data_in:in STD_LOGIC_VECTOR (An_in_ws-1 downto 0);
23        fb_scale:in STD_LOGIC_VECTOR (3 downto 0);
24        fb_data_out:out STD_LOGIC_VECTOR (Sy_out_ws-1 downto 0));
25 end fb_n;
26
27
28 architecture fb_n of fb_n is
29
30    signal An_data_out_real:An_out_data;
31    signal An_data_out_imag:An_out_data;
32    signal Sy_data_in_real:Sy_in_data;
33    signal Sy_data_in_imag:Sy_in_data;
34    signal scaler_rdy_in:STD_LOGIC;
35    signal Sy_rdy_in:STD_LOGIC;
36    signal Rd_not:Rd_not_type;
37    signal Wr_not:Wr_not_type;
38    signal Adr_An:Adr_An_type;
39    signal Adr_Sy:Adr_Sy_type;
40    signal downsampl_del:STD_LOGIC;
41
42    component An_n is
43       port(
44          clk :in STD_LOGIC;
45          reset: in STD_LOGIC;
46          An_data_in:in STD_LOGIC_VECTOR (An_in_ws-1 downto 0);
47          An_data_out_real:out An_out_data;
48          An_data_out_imag:out An_out_data;
49          An_rdy_out:out std_logic;
50          Rd_not:in Rd_not_type;
51          Adr_An:in Adr_An_type);
52    end component;
53
54    component Sy_n is
55       port(
56          clk :in STD_LOGIC;
57          reset: in STD_LOGIC;
```

```
58          Sy_rdy_in:in std_logic;
59          Sy_data_in_real:in Sy_in_data;
60          Sy_data_in_imag:in Sy_in_data;
61          Sy_data_out:out STD_LOGIC_VECTOR (Sy_out_ws-1 downto 0);
62          Wr_not:in Wr_not_type;
63          Adr_Sy:in Adr_Sy_type;
64          downsampl_del:in STD_LOGIC);
65      end component;
66
67
68      begin
69
70      process(clk)
71         begin
72            L1:for i in 0 to nmb_of_subband/2 loop
73                  Adr_An(i) <= '1';
74                  Adr_Sy(i) <= '1';
75                  Rd_not(i) <= '0';
76                  Wr_not(i) <= clk;
77               end loop L1;
78            downsampl_del <= '0';
79      end process;
80
81
82      An_n_1: An_n port map(
83         clk => clk,
84         reset => reset,
85         An_data_in => fb_data_in,
86         An_data_out_real => An_data_out_real,
87         An_data_out_imag => An_data_out_imag,
88         An_rdy_out => Sy_rdy_in,
89         Rd_not => Rd_not,
90         Adr_An => Adr_An);
91
92
93      Sy_n_1:Sy_n port map(
94         clk => clk,
95         reset => reset,
96         Sy_rdy_in => Sy_rdy_in,
97         Sy_data_in_real => Sy_data_in_real,
98         Sy_data_in_imag => Sy_data_in_imag,
99         Sy_data_out => fb_data_out,
100        Wr_not => Wr_not,
101        Adr_Sy => Adr_Sy,
102        downsampl_del => downsampl_del);
103
104     process(An_data_out_real,An_data_out_imag)
105        begin
106        L1: for n in 0 to nmb_of_subband/2 loop
107               Sy_data_in_real(n)
108                  <= An_data_out_real(n)(16+log(nmb_of_subband)
109                     downto 9+log(nmb_of_subband));
110                  --<= An_data_out_real(n)(17 downto 10);
111               Sy_data_in_imag(n)
112                  <= An_data_out_imag(n)(16+log(nmb_of_subband)
113                     downto 9+log(nmb_of_subband));
114                  --<= An_data_out_imag(n)(17 downto 10);
```

```
115             --This scaling throws away the highest bits and the lowest bits.
116             --Only appropriate for testing. Shouldn't be implemented !!!
117             end loop L1;
118         end process;
119
120 end fb_n;
```

```
1  ------------------------------------------------------------------------
2  ----------------------an_n.vhd------------------------------------------
3  ------------------------------------------------------------------------
4  --This file describes the analysis filter bank
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14 entity An_n is
15   port(
16        clk :in STD_LOGIC;
17        reset: in STD_LOGIC;
18        An_data_in:in STD_LOGIC_VECTOR (An_in_ws-1 downto 0);
19        An_data_out_real:out An_out_data;
20        An_data_out_imag:out An_out_data;
21        An_rdy_out:out std_logic;
22        Rd_not:in Rd_not_type;
23        Adr_An:in Adr_An_type);
24 end An_n;
25
26
27
28 architecture An_n of An_n is
29
30 signal Com_An_data_in:std_logic_vector(Com_An_ws-1 downto 0);
31 signal fft_data_in_real,fft_data_in_imag:fft_in_data;
32 signal fft_data_out_real,fft_data_out_imag:fft_out_data;
33 signal An_fil_data_out:An_fil_out_data;
34 signal Upsby2_data_in:Com_An_out_data;
35 signal Upsby2_data_out:Com_An_out_data;
36 signal An_fil_data_in:An_fil_in_data;
37 signal rdy_in:STD_LOGIC;
38 signal fft_rdy_out:STD_LOGIC;
39 signal An_out_reg_real:An_out_data;
40 signal An_out_reg_imag:An_out_data;
41
42    component com_an_n port(
43       Clk:in STD_LOGIC;
44       Reset:in STD_LOGIC;
45       Com_An_data_in:in STD_LOGIC_VECTOR (Com_An_ws-1 downto 0);
46       Com_An_data_out:out Com_An_out_data);
47    end component;
48
49    component upsby2_n port(
50       Clk:in STD_LOGIC;
51       Reset:in STD_LOGIC;
52       Upsby2_data_in:in Com_An_out_data;
53       Upsby2_data_out:out Com_An_out_data);
54    end component;
55
56    component an_fil port(
57       Clk:in STD_LOGIC;
```

```
58          Reset:in STD_LOGIC;
59          An_fil_data_in:in An_fil_in_data;
60          An_fil_data_out:out An_fil_out_data;
61          An_fil_rdy_out:out STD_LOGIC);
62       end component;
63
64       component An_fil8 port(
65          Clk:in STD_LOGIC;
66          Reset:in STD_LOGIC;
67          An_fil_data_in:in An_fil_in_data;
68          An_fil_data_out:out An_fil_out_data;
69          An_fil_rdy_out:out STD_LOGIC);
70       end component;
71
72       component An_fil16 port(
73          Clk:in STD_LOGIC;
74          Reset:in STD_LOGIC;
75          An_fil_data_in:in An_fil_in_data;
76          An_fil_data_out:out An_fil_out_data;
77          An_fil_rdy_out:out STD_LOGIC);
78       end component;
79
80
81       component fft port(
82          clk:in std_logic;
83          rdy_in:in std_logic;
84          data_in_real:in fft_in_data;
85          data_in_imag:in fft_in_data;
86          data_out_real:out fft_out_data;
87          data_out_imag:out fft_out_data;
88          rdy_out:out std_logic);
89       end component;
90
91
92       begin
93
94       process(An_data_in)
95          begin  --this process only type conversion
96             Com_An_data_in
97                <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER(An_data_in),Com_An_ws);
98       end process;
99
100      com_an_n_1:com_an_n port map(
101         clk => clk,
102         Reset => reset,
103         Com_An_data_in => Com_An_data_in,
104         Com_An_data_out => Upsby2_data_in);
105
106      upsby2_n_1:upsby2_n port map(
107         Clk => clk,
108         Reset => reset,
109         Upsby2_data_in => Upsby2_data_in,
110         Upsby2_data_out => Upsby2_data_out);
111
112      process(Upsby2_data_out)
113         begin  --this process only type conversion
114            L1:for i in 0 to nmb_of_FFT_pt-1 loop
```

```
115              An_fil_data_in(i) <= CONV_STD_LOGIC_VECTOR
116                 (CONV_INTEGER(Upsby2_data_out(i)),An_filt_in_ws);
117           end loop L1;
118     end process;
119
120  G1:if Nmb_of_Subband = 4 generate
121     an_fil_1:an_fil port map(
122        Clk => clk,
123        Reset => reset,
124        An_fil_data_in => An_fil_data_in,
125        An_fil_data_out => An_fil_data_out,
126        An_fil_rdy_out => rdy_in);
127     end generate G1;
128
129  G2:if Nmb_of_Subband = 8 generate
130     an_fil_1:an_fil8 port map(
131        Clk => clk,
132        Reset => reset,
133        An_fil_data_in => An_fil_data_in,
134        An_fil_data_out => An_fil_data_out,
135        An_fil_rdy_out => rdy_in);
136     end generate G2;
137
138  G3:if Nmb_of_Subband = 16 generate
139     an_fil_1:an_fil16 port map(
140        Clk => clk,
141        Reset => reset,
142        An_fil_data_in => An_fil_data_in,
143        An_fil_data_out => An_fil_data_out,
144        An_fil_rdy_out => rdy_in);
145     end generate G3;
146
147
148     process(An_fil_data_out)
149     --not really necessary because wordsize of filter output
150     -- = wordsize of fft input, but it allows to use different
151     --names and downsize the fft in order to save area
152        begin
153           L1:for i in 0 to nmb_of_FFT_pt-1 loop
154              fft_data_in_real(i) <= CONV_STD_LOGIC_VECTOR
155                 (CONV_INTEGER(An_fil_data_out(i)),fft_in_ws);
156              fft_data_in_imag(i) <= CONV_STD_LOGIC_VECTOR(0,fft_in_ws);
157           end loop L1;
158     end process;
159
160
161     fft_1:fft port map(
162        clk => clk,
163        rdy_in => rdy_in,
164        data_in_real => fft_data_in_real,
165        data_in_imag => fft_data_in_imag,
166        data_out_real => fft_data_out_real,
167        data_out_imag => fft_data_out_imag,
168        rdy_out => fft_rdy_out);
169
170
171     process(clk)  --registering fft result, type conversion
```

```
172         begin
173            if (Clk'event and Clk = '1') then
174               if (fft_rdy_out = '1') then
175                  L1:for i in 0 to nmb_of_subband/2 loop
176                        An_out_reg_real(i) <= CONV_STD_LOGIC_VECTOR
177                           (CONV_INTEGER(fft_data_out_real(i)),An_out_ws);
178                        An_out_reg_imag(i) <= CONV_STD_LOGIC_VECTOR
179                           (CONV_INTEGER(fft_data_out_imag(i)),An_out_ws);
180                     end loop L1;
181               end if;
182            end if;
183      end process;
184
185      process(clk,reset)  --pass rdy signal throgh, pull it up during reset
186         begin
187            if (reset = '0') then
188               An_rdy_out <= '1';
189            elsif (Clk'event and Clk = '1') then
190                  An_rdy_out <= fft_rdy_out;
191            end if;
192      end process;
193
194
195      process(Rd_not,Adr_An,An_out_reg_real,An_out_reg_imag)  --tristate buffer
196         begin
197            L1:for i in 0 to nmb_of_subband/2 loop
198                  if ((Rd_not(i) = '0') AND (Adr_An(i) = '1')) then
199                     An_data_out_real(i) <= An_out_reg_real(i);
200                     An_data_out_imag(i) <= An_out_reg_imag(i);
201                  else
202                     An_data_out_real(i) <=
203   CONV_STD_LOGIC_VECTOR('Z',An_out_ws);
204                     An_data_out_imag(i) <=
205   CONV_STD_LOGIC_VECTOR('Z',An_out_ws);
206                  end if;
207               end loop L1;
208      end process;
209
210   end An_n;
```

```
1  ------------------------------------------------------------------------
2  ----------------------Com_An.vhd----------------------------------------
3  ------------------------------------------------------------------------
4  --This file describes the commutator of the analysis filter bank
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library Work;
12 use Work.fltb_par.all;
13
14 entity Com_An_n is
15    port(
16       Clk:in STD_LOGIC;
17       Reset:in STD_LOGIC;
18       Com_An_data_in:in STD_LOGIC_VECTOR (Com_An_ws-1 downto 0);
19       Com_An_data_out:out Com_An_out_data);
20 end Com_An_n;
21
22 architecture Com_An_n_arch of Com_An_n is
23
24    signal one_hot_cnt:std_logic_vector(15 downto 0);
25    signal outp_reg:Com_An_out_data;
26
27    begin
28
29    process(Clk,Reset)
30    begin
31       if Reset = '0' then
32          one_hot_cnt <= "0000000000000001";
33       elsif (Clk'event and Clk = '1') then
34          one_hot_cnt(15 downto 1) <= one_hot_cnt(14 downto 0);
35          one_hot_cnt(0) <= one_hot_cnt(15);
36       end if;
37    end process;
38
39    process(Clk)
40    begin
41       if (Clk'event and Clk = '1') then
42       L1:for i in 0 to Nmb_of_subband-1 loop
43 --       if (Clk'event and Clk = '1') then
44             if one_hot_cnt(16/Nmb_of_subband*i) = '1' then
45                outp_reg(i) <= Com_An_data_in;
46             end if;
47 --         end if;
48          end loop L1;
49       end if;
50    end process;
51
52    process(outp_reg)
53    begin
54       Com_An_data_out <= outp_reg;
55    end process;
56
57 end Com_An_n_arch;
```

```
1  ------------------------------------------------------------------------
2  ----------------------upsby2_n.vhd-------------------------------------
3  ------------------------------------------------------------------------
4  --This file describes the upsampler of the analysis filter bank
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library Work;
12 use Work.fltb_par.all;
13
14 entity upsby2_n is
15   port(
16      Clk:in STD_LOGIC;
17      Reset:in STD_LOGIC;
18      Upsby2_data_in:in Com_An_out_data;
19      Upsby2_data_out:out Com_An_out_data);
20 end upsby2_n;
21
22 architecture upsby2_n_arch of upsby2_n is
23
24    signal cnt:std_logic_vector(3 downto 0);
25    signal reg:Com_An_out_data;
26
27    begin
28
29    process(Clk,Reset)
30    begin
31       if Reset = '0' then
32          cnt <= "0000";
33          L1:for i in 0 to Nmb_of_subband-1 loop
34             reg(i) <= CONV_STD_LOGIC_VECTOR(0,Com_An_ws);
35          end loop L1;
36          elsif CLK'event and CLK ='1' then
37             cnt <= cnt + 1;
38             if cnt = "0000" then
39                reg <= Upsby2_data_in;
40             elsif cnt = "1000" then
41                L2:for i in 0 to Nmb_of_subband-1 loop
42                   reg(i) <= CONV_STD_LOGIC_VECTOR(0,Com_An_ws);
43                end loop L2;
44             else reg <= reg;
45             end if;
46       end if;
47    end process;
48
49    process(reg)
50    begin
51       Upsby2_data_out <= reg;
52    end process;
53
54 end upsby2_n_arch;
55
56
```

```
1  ----------------------------------------------------------------------------
2  --------------------------------an_fil4.vhd-------------------------------
3  ----------------------------------------------------------------------------
4  --This file puts 4 filters, generated by CORE GENERATOR together
5  --Similar files are created for the 8 and 16 subband analyzer
6
7  library IEEE;
8  use IEEE.STD_LOGIC_1164.all;
9  use IEEE.STD_LOGIC_arith.all;
10 use IEEE.STD_LOGIC_signed.all;
11
12 library Work;
13 use Work.fltb_par.all;
14
15 entity An_fil is
16   port(
17      Clk:in STD_LOGIC;
18      Reset:in STD_LOGIC;
19      An_fil_data_in:in An_fil_in_data;
20      An_fil_data_out:out An_fil_out_data;
21      An_fil_rdy_out:out STD_LOGIC);
22 end An_fil;
23
24 architecture An_fil_arch of An_fil is
25
26 component an4_pp1 port (
27       data: IN std_logic_VECTOR(An_filt_in_ws-1 downto 0);
28       nd: IN std_logic;
29       rfd: OUT std_logic;
30       sinf: IN std_logic;
31       sinr: IN std_logic;
32       soutf: OUT std_logic;
33       soutr: OUT std_logic;
34       ck: IN std_logic;
35       rslt: OUT std_logic_VECTOR(An_filt_out_ws-1 downto 0);
36       rdy: OUT std_logic);
37 end component;
38
39 component an4_pp2 port (
40       data: IN std_logic_VECTOR(An_filt_in_ws-1 downto 0);
41       nd: IN std_logic;
42       rfd: OUT std_logic;
43       sinf: IN std_logic;
44       sinr: IN std_logic;
45       soutf: OUT std_logic;
46       soutr: OUT std_logic;
47       ck: IN std_logic;
48       rslt: OUT std_logic_VECTOR(An_filt_out_ws-1 downto 0);
49       rdy: OUT std_logic);
50 end component;
51
52 component an4_pp3 port (
53       data: IN std_logic_VECTOR(An_filt_in_ws-1 downto 0);
54       nd: IN std_logic;
55       rfd: OUT std_logic;
56       sinf: IN std_logic;
57       sinr: IN std_logic;
```

```
58          soutf: OUT std_logic;
59          soutr: OUT std_logic;
60          ck: IN std_logic;
61          rslt: OUT std_logic_VECTOR(An_filt_out_ws-1 downto 0);
62          rdy: OUT std_logic);
63    end component;
64
65    component an4_pp4 port (
66          data: IN std_logic_VECTOR(An_filt_in_ws-1 downto 0);
67          nd: IN std_logic;
68          rfd: OUT std_logic;
69          sinf: IN std_logic;
70          sinr: IN std_logic;
71          soutf: OUT std_logic;
72          soutr: OUT std_logic;
73          ck: IN std_logic;
74          rslt: OUT std_logic_VECTOR(An_filt_out_ws-1 downto 0);
75          rdy: OUT std_logic);
76    end component;
77
78
79
80       signal nd,rfd1,rfd2,rfd3,rfd4:std_logic;
81       signal sinf1,sinf2,sinf3,sinf4,sinr1,sinr2,sinr3,sinr4:std_logic;
82       signal soutf1,soutf2,soutf3,soutf4,soutr1,soutr2,soutr3,soutr4:std_logic;
83       signal d1,d2,d3:std_logic;
84       signal filt1_in,filt2_in,filt3_in,filt4_in:
85          STD_LOGIC_VECTOR(An_filt_in_ws-1 downto 0);
86       signal filt1_out,filt2_out,filt3_out,filt4_out:
87          STD_LOGIC_VECTOR(An_filt_out_ws-1 downto 0);
88
89       begin
90
91    an4_pp1_1 : an4_pp1 port map (
92          data => filt1_in,
93          nd => nd,
94          rfd => rfd1,
95          sinf => sinf1,
96          sinr => sinr1,
97          soutf => soutf1,
98          soutr => soutr1,
99          ck => Clk,
100         rslt => filt1_out,
101         rdy => An_fil_rdy_out);
102
103   an4_pp2_1 : an4_pp2 port map (
104         data => filt2_in,
105         nd => nd,
106         rfd => rfd2,
107         sinf => sinf2,
108         sinr => sinr2,
109         soutf => soutf2,
110         soutr => soutr2,
111         ck => Clk,
112         rslt => filt2_out,
113         rdy => d1);
114
```

```
115   an4_pp3_1 : an4_pp3 port map (
116         data => filt3_in,
117         nd => nd,
118         rfd => rfd3,
119         sinf => sinf3,
120         sinr => sinr3,
121         soutf => soutf3,
122         soutr => soutr3,
123         ck => Clk,
124         rslt => filt3_out,
125         rdy => d2);
126
127   an4_pp4_1 : an4_pp4 port map (
128         data => filt4_in,
129         nd => nd,
130         rfd => rfd4,
131         sinf => sinf4,
132         sinr => sinr4,
133         soutf => soutf4,
134         soutr => soutr4,
135         ck => Clk,
136         rslt => filt4_out,
137         rdy =>d3);
138
139
140
141      process(rfd1,rfd2,rfd3,rfd4)
142      begin
143         nd <= rfd1 and rfd2 and rfd3 and rfd4;
144      end process;
145
146      process(An_fil_data_in,filt1_out,filt2_out,filt3_out,filt4_out)
147      begin
148         filt1_in <= An_fil_data_in(0);
149         filt2_in <= An_fil_data_in(1);
150         filt3_in <= An_fil_data_in(2);
151         filt4_in <= An_fil_data_in(3);
152         An_fil_data_out(0) <= filt1_out;
153         An_fil_data_out(1) <= filt2_out;
154         An_fil_data_out(2) <= filt3_out;
155         An_fil_data_out(3) <= filt4_out;
156      end process;
157
158   end An_fil_arch;
159
160
```

```
1  --------------------------------------------------------------------------
2  ------------------------------fft.vhd-------------------------------------
3  --------------------------------------------------------------------------
4  --This file describes the fft, used in the analysis filter bank
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14
15 entity fft is
16   port(
17       clk:in std_logic;
18       rdy_in:in std_logic;
19       data_in_real:in fft_in_data;
20       data_in_imag:in fft_in_data;
21       data_out_real:out fft_out_data;
22       data_out_imag:out fft_out_data;
23       rdy_out:out std_logic);
24 end fft;
25
26
27
28 architecture fft_arch of fft is
29
30 component fft_inp_reg
31     generic(
32         wordsize:integer);
33     port(
34         SR_clk:in std_logic;
35         SR_rdy_in:in std_logic;
36         SR_input_real:in fft_in_data;          --array
37         SR_input_imag:in fft_in_data;          --array
38         SR_data_out_0_real:out std_logic_vector(wordsize-1 downto 0);
39         SR_data_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
40         SR_data_out_1_real:out std_logic_vector(wordsize-1 downto 0);
41         SR_data_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
42         SR_ctrl_cnt_out:out std_logic_vector(3 downto 0));
43 end component;
44
45 component butterfly
46   generic(
47       wordsize:integer);
48   port(
49       Btf_clk:in std_logic;
50       Btf_ctrl_cnt_in:in std_logic_vector(3 downto 0);
51       Btf_in_0_real:in std_logic_vector(wordsize-1 downto 0);
52       Btf_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
53       Btf_in_1_real:in std_logic_vector(wordsize-1 downto 0);
54       Btf_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
55       Btf_out_0_real:out std_logic_vector(wordsize downto 0);
56       Btf_out_0_imag:out std_logic_vector(wordsize downto 0);
57       Btf_out_1_real:out std_logic_vector(wordsize downto 0);
```

```
58          Btf_out_1_imag:out std_logic_vector(wordsize downto 0);
59          Btf_ctrl_cnt_out:out std_logic_vector(3 downto 0));
60    end component;
61
62    component delay
63      generic(
64          stage:integer;
65          delayed_path:integer;
66          wordsize:integer);
67      port(
68          Del_clk:in std_logic;
69          Del_ctrl_cnt_in:in std_logic_vector(3 downto 0);
70          Del_data_in_0_real:in std_logic_vector(wordsize-1 downto 0);
71          Del_data_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
72          Del_data_in_1_real:in std_logic_vector(wordsize-1 downto 0);
73          Del_data_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
74          Del_data_out_0_real:out std_logic_vector(wordsize-1 downto 0);
75          Del_data_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
76          Del_data_out_1_real:out std_logic_vector(wordsize-1 downto 0);
77          Del_data_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
78          Del_ctrl_cnt_out:out std_logic_vector(3 downto 0));
79    end component;
80
81    component fft_comm
82      generic(
83          wordsize:integer;
84          controlbit:integer);  --0 to 3, determines which bit of the control
85                                --counter switches the commutator
86      port(
87          Com_clk:in std_logic;
88          Com_ctrl_cnt_in:in std_logic_vector(3 downto 0);
89          Com_in_0_real:in std_logic_vector(wordsize-1 downto 0);
90          Com_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
91          Com_in_1_real:in std_logic_vector(wordsize-1 downto 0);
92          Com_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
93          Com_out_0_real:out std_logic_vector(wordsize-1 downto 0);
94          Com_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
95          Com_out_1_real:out std_logic_vector(wordsize-1 downto 0);
96          Com_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
97          Com_ctrl_cnt_out:out std_logic_vector(3 downto 0));
98    end component;
99
100   component mul2val
101     generic(
102         wordsize:integer;
103         controlbit:integer);  --0 to 3, determines which bit of the control
104                               --counter selects the factor (1 or j)
105     port(
106         M2_clk:in std_logic;
107         M2_ctrl_cnt_in:in std_logic_vector(3 downto 0);
108         M2_in_0_real:in std_logic_vector(wordsize-1 downto 0);
109         M2_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
110         M2_in_1_real:in std_logic_vector(wordsize-1 downto 0);
111         M2_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
112         M2_out_0_real:out std_logic_vector(wordsize-1 downto 0);
113         M2_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
114         M2_out_1_real:out std_logic_vector(wordsize-1 downto 0);
```

```
115         M2_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
116         M2_ctrl_cnt_out:out std_logic_vector(3 downto 0));
117   end component;
118
119   component mul4val
120     generic(
121         ws:integer;  --wordsize
122         ctrl_word_LSB:integer);
123     port(
124         M4_clk:in std_logic;
125         M4_ctrl_cnt_in:in std_logic_vector(3 downto 0);
126         M4_in_0_real:in std_logic_vector(ws-1 downto 0);
127         M4_in_0_imag:in std_logic_vector(ws-1 downto 0);
128         M4_in_1_real:in std_logic_vector(ws-1 downto 0);
129         M4_in_1_imag:in std_logic_vector(ws-1 downto 0);
130         M4_out_0_real:out std_logic_vector(ws-1 downto 0);
131         M4_out_0_imag:out std_logic_vector(ws-1 downto 0);
132         M4_out_1_real:out std_logic_vector(ws-1 downto 0);
133         M4_out_1_imag:out std_logic_vector(ws-1 downto 0);
134         M4_ctrl_cnt_out:out std_logic_vector(3 downto 0));
135   end component;
136
137   component mul8val
138     generic(
139         ws:integer;  --wordsize
140         ctrl_word_LSB:integer);
141     port(
142         M8_clk:in std_logic;
143         M8_ctrl_cnt_in:in std_logic_vector(3 downto 0);
144         M8_in_0_real:in std_logic_vector(ws-1 downto 0);
145         M8_in_0_imag:in std_logic_vector(ws-1 downto 0);
146         M8_in_1_real:in std_logic_vector(ws-1 downto 0);
147         M8_in_1_imag:in std_logic_vector(ws-1 downto 0);
148         M8_out_0_real:out std_logic_vector(ws-1 downto 0);
149         M8_out_0_imag:out std_logic_vector(ws-1 downto 0);
150         M8_out_1_real:out std_logic_vector(ws-1 downto 0);
151         M8_out_1_imag:out std_logic_vector(ws-1 downto 0);
152         M8_ctrl_cnt_out:out std_logic_vector(3 downto 0));
153   end component;
154
155   component fft_out_reg
156     generic(
157         wordsize:integer);
158     port(
159         SR_clk:in std_logic;
160         SR_ctrl_cnt_in:in std_logic_vector(3 downto 0);
161         SR_data_in_0_real:in std_logic_vector(wordsize-1 downto 0);
162         SR_data_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
163         SR_data_in_1_real:in std_logic_vector(wordsize-1 downto 0);
164         SR_data_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
165         SR_rdy_out:out std_logic;
166         SR_output_real:out fft_out_data;          --array
167         SR_output_imag:out fft_out_data);          --array
168   end component;
169
170   ----------SIGNALS FOR STAGE 0-----------------------------------------------
171   ----------input shift register signals
```

```
172   signal inp_SR_out_0_r,inp_SR_out_0_i,inp_SR_out_1_r,inp_SR_out_1_i:
173      std_logic_vector(fft_in_ws-1 downto 0);
174   signal inp_SR_ctrl_cnt_out:std_logic_vector(3 downto 0);
175   ----------signals from butterfly #0
176   signal bf_0_ctrl_cnt_out:std_logic_vector(3 downto 0);
177   signal bf_0_out_0_r,bf_0_out_0_i,bf_0_out_1_r,bf_0_out_1_i:
178      std_logic_vector(fft_in_ws-1+1 downto 0);
179   ----------SIGNALS FOR STAGE 1-----------------------------------------
180   ----------signals from delay #0
181   signal del_0_ctrl_cnt_out:std_logic_vector(3 downto 0);
182   signal del_0_out_0_r,del_0_out_0_i,del_0_out_1_r,del_0_out_1_i:
183      std_logic_vector(fft_in_ws-1+1 downto 0);
184   ----------signals from commutator #0
185   signal com_0_ctrl_cnt_out:std_logic_vector(3 downto 0);
186   signal com_0_out_0_r,com_0_out_0_i,com_0_out_1_r,com_0_out_1_i:
187      std_logic_vector(fft_in_ws-1+1 downto 0);
188   ----------signals from delay #1
189   signal del_1_ctrl_cnt_out:std_logic_vector(3 downto 0);
190   signal del_1_out_0_r,del_1_out_0_i,del_1_out_1_r,del_1_out_1_i:
191      std_logic_vector(fft_in_ws-1+1 downto 0);
192   ----------signals from multiplier #0
193   signal mul_0_ctrl_cnt_out:std_logic_vector(3 downto 0);
194   signal mul_0_out_0_r,mul_0_out_0_i,mul_0_out_1_r,mul_0_out_1_i:
195      std_logic_vector(fft_in_ws-1+1 downto 0);
196   ----------signals from butterfly #1
197   signal bf_1_ctrl_cnt_out:std_logic_vector(3 downto 0);
198   signal bf_1_out_0_r,bf_1_out_0_i,bf_1_out_1_r,bf_1_out_1_i:
199      std_logic_vector(fft_in_ws-1+2 downto 0);
200   ----------SIGNALS FOR STAGE 2-----------------------------------------
201   ----------signals from delay #2
202   signal del_2_ctrl_cnt_out:std_logic_vector(3 downto 0);
203   signal del_2_out_0_r,del_2_out_0_i,del_2_out_1_r,del_2_out_1_i:
204      std_logic_vector(fft_in_ws-1+2 downto 0);
205   ----------signals from commutator #1
206   signal com_1_ctrl_cnt_out:std_logic_vector(3 downto 0);
207   signal com_1_out_0_r,com_1_out_0_i,com_1_out_1_r,com_1_out_1_i:
208      std_logic_vector(fft_in_ws-1+2 downto 0);
209   ----------signals from delay #3
210   signal del_3_ctrl_cnt_out:std_logic_vector(3 downto 0);
211   signal del_3_out_0_r,del_3_out_0_i,del_3_out_1_r,del_3_out_1_i:
212      std_logic_vector(fft_in_ws-1+2 downto 0);
213   ----------signals from multiplier #1
214   signal mul_1_ctrl_cnt_out:std_logic_vector(3 downto 0);
215   signal mul_1_out_0_r,mul_1_out_0_i,mul_1_out_1_r,mul_1_out_1_i:
216      std_logic_vector(fft_in_ws-1+2 downto 0);
217   ----------signals from butterfly #2
218   signal bf_2_ctrl_cnt_out:std_logic_vector(3 downto 0);
219   signal bf_2_out_0_r,bf_2_out_0_i,bf_2_out_1_r,bf_2_out_1_i:
220      std_logic_vector(fft_in_ws-1+3 downto 0);
221   ----------SIGNALS FOR STAGE 3-----------------------------------------
222   ----------signals from delay #4
223   signal del_4_ctrl_cnt_out:std_logic_vector(3 downto 0);
224   signal del_4_out_0_r,del_4_out_0_i,del_4_out_1_r,del_4_out_1_i:
225      std_logic_vector(fft_in_ws-1+3 downto 0);
226   ----------signals from commutator #2
227   signal com_2_ctrl_cnt_out:std_logic_vector(3 downto 0);
228   signal com_2_out_0_r,com_2_out_0_i,com_2_out_1_r,com_2_out_1_i:
```

```
229      std_logic_vector(fft_in_ws-1+3 downto 0);
230   ----------signals from delay #5
231   signal del_5_ctrl_cnt_out:std_logic_vector(3 downto 0);
232   signal del_5_out_0_r,del_5_out_0_i,del_5_out_1_r,del_5_out_1_i:
233      std_logic_vector(fft_in_ws-1+3 downto 0);
234   ----------signals from multiplier #2
235   signal mul_2_ctrl_cnt_out:std_logic_vector(3 downto 0);
236   signal mul_2_out_0_r,mul_2_out_0_i,mul_2_out_1_r,mul_2_out_1_i:
237      std_logic_vector(fft_in_ws-1+3 downto 0);
238   ----------signals from butterfly #3
239   signal bf_3_ctrl_cnt_out:std_logic_vector(3 downto 0);
240   signal bf_3_out_0_r,bf_3_out_0_i,bf_3_out_1_r,bf_3_out_1_i:
241      std_logic_vector(fft_in_ws-1+4 downto 0);
242
243
244
245       begin
246       fft_inp_reg_1:fft_inp_reg
247         generic map(
248           wordsize => fft_in_ws)
249         port map(
250           SR_clk => clk,
251           SR_rdy_in => rdy_in,
252           SR_input_real => data_in_real,
253           SR_input_imag => data_in_imag,
254           SR_data_out_0_real => inp_SR_out_0_r,
255           SR_data_out_0_imag => inp_SR_out_0_i,
256           SR_data_out_1_real => inp_SR_out_1_r,
257           SR_data_out_1_imag => inp_SR_out_1_i,
258           SR_ctrl_cnt_out => inp_SR_ctrl_cnt_out);
259
260   --stage 0 just one butterfly, always required except for trivial (1 pt FFT)
261    G0:if log(nmb_of_fft_pt) > 0 generate
262       butterfly_0:butterfly
263           generic map(
264               wordsize => fft_in_ws)
265           port map(
266               Btf_clk => clk,
267                     Btf_ctrl_cnt_in => inp_SR_ctrl_cnt_out,
268                     Btf_in_0_real => inp_SR_out_0_r,
269                     Btf_in_0_imag => inp_SR_out_0_i,
270                     Btf_in_1_real => inp_SR_out_1_r,
271                     Btf_in_1_imag => inp_SR_out_1_i,
272                     Btf_out_0_real => bf_0_out_0_r,
273                     Btf_out_0_imag => bf_0_out_0_i,
274                     Btf_out_1_real => bf_0_out_1_r,
275                     Btf_out_1_imag => bf_0_out_1_i,
276                     Btf_ctrl_cnt_out => bf_0_ctrl_cnt_out);
277       end generate;
278   --stage 1, neccessary if number of FFT pts > 2----------------------
279    G1:if log(nmb_of_FFT_pt)-1 > 0 generate
280
281       delay_0:delay
282           generic map(
283               stage => 2**(abs(log(nmb_of_FFT_pt)-2)),--abs only to make
284                                                       --compiler quiet
285               delayed_path => 1,
```

```
286             wordsize => fft_in_ws+1)
287         port map(
288             Del_clk => clk,
289             Del_ctrl_cnt_in => bf_0_ctrl_cnt_out,
290                   Del_data_in_0_real => bf_0_out_0_r,
291                   Del_data_in_0_imag => bf_0_out_0_i,
292                   Del_data_in_1_real => bf_0_out_1_r,
293                   Del_data_in_1_imag => bf_0_out_1_i,
294                   Del_data_out_0_real => del_0_out_0_r,
295                   Del_data_out_0_imag => del_0_out_0_i,
296                   Del_data_out_1_real => del_0_out_1_r,
297                   Del_data_out_1_imag => del_0_out_1_i,
298                   Del_ctrl_cnt_out => del_0_ctrl_cnt_out);
299
300     fft_comm_0:fft_comm
301         generic map(
302             wordsize => fft_in_ws+1,
303             controlbit => log(nmb_of_FFT_pt)-2)  --0 to 3, determines
304             --which bit of the control counter switches the commutator
305         port map(
306             Com_clk => clk,
307             Com_ctrl_cnt_in => del_0_ctrl_cnt_out,
308                   Com_in_0_real => del_0_out_0_r,
309             Com_in_0_imag => del_0_out_0_i,
310                   Com_in_1_real => del_0_out_1_r,
311                   Com_in_1_imag => del_0_out_1_i,
312                   Com_out_0_real => com_0_out_0_r,
313                   Com_out_0_imag => com_0_out_0_i,
314                   Com_out_1_real => com_0_out_1_r,
315                   Com_out_1_imag => com_0_out_1_i,
316                   Com_ctrl_cnt_out => com_0_ctrl_cnt_out);
317
318     delay_1:delay
319         generic map(
320             stage => 2**(abs(log(nmb_of_FFT_pt)-2)),
321             delayed_path => 0,
322             wordsize => fft_in_ws+1)
323         port map(
324             Del_clk => clk,
325             Del_ctrl_cnt_in => com_0_ctrl_cnt_out,
326                   Del_data_in_0_real => com_0_out_0_r,
327                   Del_data_in_0_imag => com_0_out_0_i,
328                   Del_data_in_1_real => com_0_out_1_r,
329                   Del_data_in_1_imag => com_0_out_1_i,
330                   Del_data_out_0_real => del_1_out_0_r,
331                   Del_data_out_0_imag => del_1_out_0_i,
332                   Del_data_out_1_real => del_1_out_1_r,
333                   Del_data_out_1_imag => del_1_out_1_i,
334                   Del_ctrl_cnt_out => del_1_ctrl_cnt_out);
335
336     mul2val_0:mul2val
337         generic map(
338             wordsize => fft_in_ws+1,
339             controlbit => log(nmb_of_FFT_pt)-2)  --0 to 3,
340             --determines which bit of the control counter selects the factor
341 (1 or j)
342         port map(
```

```
343                 M2_clk => clk,
344                       M2_ctrl_cnt_in => del_1_ctrl_cnt_out,
345                       M2_in_0_real => del_1_out_0_r,
346                       M2_in_0_imag => del_1_out_0_i,
347                       M2_in_1_real => del_1_out_1_r,
348                       M2_in_1_imag => del_1_out_1_i,
349                       M2_out_0_real => mul_0_out_0_r,
350                       M2_out_0_imag => mul_0_out_0_i,
351                       M2_out_1_real => mul_0_out_1_r,
352                       M2_out_1_imag => mul_0_out_1_i,
353                       M2_ctrl_cnt_out => mul_0_ctrl_cnt_out);
354
355       butterfly_1:butterfly
356           generic map(
357               wordsize => fft_in_ws+1)
358           port map(
359               Btf_clk => clk,
360                       Btf_ctrl_cnt_in => mul_0_ctrl_cnt_out,
361                       Btf_in_0_real => mul_0_out_0_r,
362                       Btf_in_0_imag => mul_0_out_0_i,
363                       Btf_in_1_real => mul_0_out_1_r,
364                       Btf_in_1_imag => mul_0_out_1_i,
365                       Btf_out_0_real => bf_1_out_0_r,
366                       Btf_out_0_imag => bf_1_out_0_i,
367                       Btf_out_1_real => bf_1_out_1_r,
368                       Btf_out_1_imag => bf_1_out_1_i,
369                       Btf_ctrl_cnt_out => bf_1_ctrl_cnt_out);
370
371       end generate;
372   --stage 2, necessary if number of FFT pts > 4-----------------------
373    G2:if log(nmb_of_FFT_pt)-2 > 0 generate
374
375       delay_2:delay
376           generic map(
377               stage => 2**(abs(log(nmb_of_FFT_pt)-3)),
378               delayed_path => 1,
379               wordsize => fft_in_ws+2)
380           port map(
381               Del_clk => clk,
382               Del_ctrl_cnt_in => bf_1_ctrl_cnt_out,
383                       Del_data_in_0_real => bf_1_out_0_r,
384                       Del_data_in_0_imag => bf_1_out_0_i,
385                       Del_data_in_1_real => bf_1_out_1_r,
386                       Del_data_in_1_imag => bf_1_out_1_i,
387                       Del_data_out_0_real => del_2_out_0_r,
388                       Del_data_out_0_imag => del_2_out_0_i,
389                       Del_data_out_1_real => del_2_out_1_r,
390                       Del_data_out_1_imag => del_2_out_1_i,
391                       Del_ctrl_cnt_out => del_2_ctrl_cnt_out);
392
393       fft_comm_1:fft_comm
394           generic map(
395               wordsize => fft_in_ws+2,
396               controlbit => log(nmb_of_FFT_pt)-3)  --0 to 3,
397               --determines which bit of the control counter switches the
398   commut.
399           port map(
```

```
400             Com_clk => clk,
401             Com_ctrl_cnt_in => del_2_ctrl_cnt_out,
402                   Com_in_0_real => del_2_out_0_r,
403             Com_in_0_imag => del_2_out_0_i,
404                   Com_in_1_real => del_2_out_1_r,
405                   Com_in_1_imag => del_2_out_1_i,
406                   Com_out_0_real => com_1_out_0_r,
407                   Com_out_0_imag => com_1_out_0_i,
408                   Com_out_1_real => com_1_out_1_r,
409                   Com_out_1_imag => com_1_out_1_i,
410                   Com_ctrl_cnt_out => com_1_ctrl_cnt_out);
411
412     delay_3:delay
413         generic map(
414             stage => 2**(abs(log(nmb_of_FFT_pt)-3)),
415             delayed_path => 0,
416             wordsize => fft_in_ws+2)
417         port map(
418             Del_clk => clk,
419             Del_ctrl_cnt_in => com_1_ctrl_cnt_out,
420                   Del_data_in_0_real => com_1_out_0_r,
421                   Del_data_in_0_imag => com_1_out_0_i,
422                   Del_data_in_1_real => com_1_out_1_r,
423                   Del_data_in_1_imag => com_1_out_1_i,
424                   Del_data_out_0_real => del_3_out_0_r,
425                   Del_data_out_0_imag => del_3_out_0_i,
426                   Del_data_out_1_real => del_3_out_1_r,
427                   Del_data_out_1_imag => del_3_out_1_i,
428                   Del_ctrl_cnt_out => del_3_ctrl_cnt_out);
429
430     mul4val_0:mul4val
431         generic map(
432             ws => fft_in_ws+2,
433             ctrl_word_LSB => log(nmb_of_FFT_pt)-3)
434         port map(
435             M4_clk => clk,
436                   M4_ctrl_cnt_in => del_3_ctrl_cnt_out,
437                   M4_in_0_real => del_3_out_0_r,
438                   M4_in_0_imag => del_3_out_0_i,
439                   M4_in_1_real => del_3_out_1_r,
440                   M4_in_1_imag => del_3_out_1_i,
441                   M4_out_0_real => mul_1_out_0_r,
442                   M4_out_0_imag => mul_1_out_0_i,
443                   M4_out_1_real => mul_1_out_1_r,
444                   M4_out_1_imag => mul_1_out_1_i,
445                   M4_ctrl_cnt_out => mul_1_ctrl_cnt_out);
446
447     butterfly_2:butterfly
448         generic map(
449             wordsize => fft_in_ws+2)
450         port map(
451             Btf_clk => clk,
452                   Btf_ctrl_cnt_in => mul_1_ctrl_cnt_out,
453                   Btf_in_0_real => mul_1_out_0_r,
454                   Btf_in_0_imag => mul_1_out_0_i,
455                   Btf_in_1_real => mul_1_out_1_r,
456                   Btf_in_1_imag => mul_1_out_1_i,
```

```
457                     Btf_out_0_real => bf_2_out_0_r,
458                     Btf_out_0_imag => bf_2_out_0_i,
459                     Btf_out_1_real => bf_2_out_1_r,
460                     Btf_out_1_imag => bf_2_out_1_i,
461                     Btf_ctrl_cnt_out => bf_2_ctrl_cnt_out);
462
463     end generate;
464  ---------------stage 3, necessary if number of FFT pts > 8----------------
465   G3:if log(nmb_of_FFT_pt)-3 > 0 generate
466
467     delay_4:delay
468         generic map(
469             stage => 2**(abs(log(nmb_of_FFT_pt)-4)),
470             delayed_path => 1,
471             wordsize => fft_in_ws+3)
472         port map(
473             Del_clk => clk,
474             Del_ctrl_cnt_in => bf_2_ctrl_cnt_out,
475                   Del_data_in_0_real => bf_2_out_0_r,
476                   Del_data_in_0_imag => bf_2_out_0_i,
477                   Del_data_in_1_real => bf_2_out_1_r,
478                   Del_data_in_1_imag => bf_2_out_1_i,
479                   Del_data_out_0_real => del_4_out_0_r,
480                   Del_data_out_0_imag => del_4_out_0_i,
481                   Del_data_out_1_real => del_4_out_1_r,
482                   Del_data_out_1_imag => del_4_out_1_i,
483                   Del_ctrl_cnt_out => del_4_ctrl_cnt_out);
484
485     fft_comm_2:fft_comm
486         generic map(
487             wordsize => fft_in_ws+3,
488             controlbit => log(nmb_of_FFT_pt)-4)  --0 to 3,
489             --determines which bit of the control counter switches the
490  commut.
491         port map(
492             Com_clk => clk,
493             Com_ctrl_cnt_in => del_4_ctrl_cnt_out,
494                   Com_in_0_real => del_4_out_0_r,
495             Com_in_0_imag => del_4_out_0_i,
496                   Com_in_1_real => del_4_out_1_r,
497                   Com_in_1_imag => del_4_out_1_i,
498                   Com_out_0_real => com_2_out_0_r,
499                   Com_out_0_imag => com_2_out_0_i,
500                   Com_out_1_real => com_2_out_1_r,
501                   Com_out_1_imag => com_2_out_1_i,
502                   Com_ctrl_cnt_out => com_2_ctrl_cnt_out);
503
504     delay_5:delay
505         generic map(
506             stage => 2**(abs(log(nmb_of_FFT_pt)-4)),
507             delayed_path => 0,
508             wordsize => fft_in_ws+3)
509         port map(
510             Del_clk => clk,
511             Del_ctrl_cnt_in => com_2_ctrl_cnt_out,
512                   Del_data_in_0_real => com_2_out_0_r,
513                   Del_data_in_0_imag => com_2_out_0_i,
```

```
514                     Del_data_in_1_real => com_2_out_1_r,
515                     Del_data_in_1_imag => com_2_out_1_i,
516                     Del_data_out_0_real => del_5_out_0_r,
517                     Del_data_out_0_imag => del_5_out_0_i,
518                     Del_data_out_1_real => del_5_out_1_r,
519                     Del_data_out_1_imag => del_5_out_1_i,
520                     Del_ctrl_cnt_out => del_5_ctrl_cnt_out);
521
522       mul8val_0:mul8val
523           generic map(
524               ws => fft_in_ws+3,
525               ctrl_word_LSB => log(nmb_of_FFT_pt)-4)
526           port map(
527               M8_clk => clk,
528                     M8_ctrl_cnt_in => del_5_ctrl_cnt_out,
529                     M8_in_0_real => del_5_out_0_r,
530                     M8_in_0_imag => del_5_out_0_i,
531                     M8_in_1_real => del_5_out_1_r,
532                     M8_in_1_imag => del_5_out_1_i,
533                     M8_out_0_real => mul_2_out_0_r,
534                     M8_out_0_imag => mul_2_out_0_i,
535                     M8_out_1_real => mul_2_out_1_r,
536                     M8_out_1_imag => mul_2_out_1_i,
537                     M8_ctrl_cnt_out => mul_2_ctrl_cnt_out);
538
539       butterfly_3:butterfly
540           generic map(
541               wordsize => fft_in_ws+3)
542           port map(
543               Btf_clk => clk,
544                     Btf_ctrl_cnt_in => mul_2_ctrl_cnt_out,
545                     Btf_in_0_real => mul_2_out_0_r,
546                     Btf_in_0_imag => mul_2_out_0_i,
547                     Btf_in_1_real => mul_2_out_1_r,
548                     Btf_in_1_imag => mul_2_out_1_i,
549                     Btf_out_0_real => bf_3_out_0_r,
550                     Btf_out_0_imag => bf_3_out_0_i,
551                     Btf_out_1_real => bf_3_out_1_r,
552                     Btf_out_1_imag => bf_3_out_1_i,
553                     Btf_ctrl_cnt_out => bf_3_ctrl_cnt_out);
554
555       end generate;
556
557   --Now decide which signal connect to the output shitf register------------
558    G4:if log(nmb_of_FFT_pt)-1 = 0 generate --2 pt FFT, just 1 butterfly
559       fft_out_reg_1:fft_out_reg
560           generic map(
561               wordsize => fft_in_ws+1)
562           port map(
563               SR_clk => clk,
564               SR_ctrl_cnt_in => bf_0_ctrl_cnt_out,
565               SR_data_in_0_real => bf_0_out_0_r,
566                     SR_data_in_0_imag => bf_0_out_0_i,
567                     SR_data_in_1_real => bf_0_out_1_r,
568                     SR_data_in_1_imag => bf_0_out_1_i,
569                     SR_rdy_out => rdy_out,
570                     SR_output_real => data_out_real,
```

```
571                     SR_output_imag => data_out_imag);
572       end generate;
573
574    G5:if log(nmb_of_FFT_pt)-2 = 0 generate
575       fft_out_reg_2:fft_out_reg
576           generic map(
577               wordsize => fft_in_ws+2)
578           port map(
579               SR_clk => clk,
580               SR_ctrl_cnt_in => bf_1_ctrl_cnt_out,
581                     SR_data_in_0_real => bf_1_out_0_r,
582                     SR_data_in_0_imag => bf_1_out_0_i,
583                     SR_data_in_1_real => bf_1_out_1_r,
584                     SR_data_in_1_imag => bf_1_out_1_i,
585                     SR_rdy_out => rdy_out,
586                     SR_output_real => data_out_real,
587                     SR_output_imag => data_out_imag);
588       end generate;
589
590    G6:if log(nmb_of_FFT_pt)-3 = 0 generate
591       fft_out_reg_3:fft_out_reg
592           generic map(
593               wordsize => fft_in_ws+3)
594           port map(
595               SR_clk => clk,
596               SR_ctrl_cnt_in => bf_2_ctrl_cnt_out,
597                     SR_data_in_0_real => bf_2_out_0_r,
598                     SR_data_in_0_imag => bf_2_out_0_i,
599                     SR_data_in_1_real => bf_2_out_1_r,
600                     SR_data_in_1_imag => bf_2_out_1_i,
601                     SR_rdy_out => rdy_out,
602                     SR_output_real => data_out_real,
603                     SR_output_imag => data_out_imag);
604       end generate;
605
606    G7:if log(nmb_of_FFT_pt)-4 = 0 generate
607       fft_out_reg_4:fft_out_reg
608           generic map(
609               wordsize => fft_in_ws+4)
610           port map(
611               SR_clk => clk,
612               SR_ctrl_cnt_in => bf_3_ctrl_cnt_out,
613                     SR_data_in_0_real => bf_3_out_0_r,
614                     SR_data_in_0_imag => bf_3_out_0_i,
615                     SR_data_in_1_real => bf_3_out_1_r,
616                     SR_data_in_1_imag => bf_3_out_1_i,
617                     SR_rdy_out => rdy_out,
618                     SR_output_real => data_out_real,
619                     SR_output_imag => data_out_imag);
620       end generate;
621
622    end fft_arch;
623
```

```
1  ----------------------------------------------------------------------------
2  ----------------------inp_sr.vhd------------------------------------------
3  ----------------------------------------------------------------------------
4  --This file describes the input shift register of the pipeline FFT
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14 entity fft_inp_reg is
15   generic(
16       wordsize:integer);
17   port(
18       SR_clk:in std_logic;
19       SR_rdy_in:in std_logic;
20       SR_input_real:in fft_in_data;        --array
21       SR_input_imag:in fft_in_data;        --array
22       SR_data_out_0_real:out std_logic_vector(wordsize-1 downto 0);
23       SR_data_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
24       SR_data_out_1_real:out std_logic_vector(wordsize-1 downto 0);
25       SR_data_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
26       SR_ctrl_cnt_out:out std_logic_vector(3 downto 0));
27 end fft_inp_reg;
28
29
30
31 architecture fft_inp_reg_arch of fft_inp_reg is
32
33 signal reg_real,reg_imag:fft_in_data;     --array
34 signal ctrl_cnt:std_logic_vector(3 downto 0);
35
36 begin
37
38
39 process(SR_clk)
40     begin
41     if SR_CLK'event and SR_CLK ='1' then
42         if SR_rdy_in = '1' then
43             reg_real <= SR_input_real;
44             reg_imag <= SR_input_imag;
45             ctrl_cnt <= "0000";
46         else
47             ctrl_cnt <= ctrl_cnt + CONV_STD_LOGIC_VECTOR(1,4);
48             if nmb_of_FFT_pt > 2 then
49                L1:for i in 0 to abs(nmb_of_FFT_pt/2-2) loop --abs only
50                   --used to get it compiled
51                   reg_real(i) <= reg_real(i+1);
52                   reg_imag(i) <= reg_imag(i+1);
53                end loop L1;
54             end if;
55             reg_real(nmb_of_FFT_pt/2-1) <=
56 CONV_STD_LOGIC_VECTOR(0,fft_in_ws);
```

```
57              reg_imag(nmb_of_FFT_pt/2-1) <=
58  CONV_STD_LOGIC_VECTOR(0,fft_in_ws);
59              L2:for i in nmb_of_FFT_pt/2 to nmb_of_FFT_pt-2 loop
60                 reg_real(i) <= reg_real(i+1);
61                 reg_imag(i) <= reg_imag(i+1);
62              end loop L2;
63              reg_real(nmb_of_FFT_pt-1) <= CONV_STD_LOGIC_VECTOR(0,fft_in_ws);
64              reg_imag(nmb_of_FFT_pt-1) <= CONV_STD_LOGIC_VECTOR(0,fft_in_ws);
65          end if;
66      end if;
67
68  end process;
69
70
71  process(reg_real,reg_imag,ctrl_cnt)
72      begin
73      SR_data_out_0_real <= reg_real(0);
74      SR_data_out_0_imag <= reg_imag(0);
75      SR_data_out_1_real <= reg_real(nmb_of_FFT_pt/2);
76      SR_data_out_1_imag <= reg_imag(nmb_of_FFT_pt/2);
77      SR_ctrl_cnt_out <= ctrl_cnt;
78  end process;
79
80
81  end fft_inp_reg_arch;
82
```

```
1   ----------------------------------------------------------------------
2   ----------------------butterfl.vhd----------------------------------
3   ----------------------------------------------------------------------
4   --This file describes the butterflies, used in the FFT/IFFT
5
6   library IEEE;
7   use IEEE.STD_LOGIC_1164.all;
8   use IEEE.STD_LOGIC_arith.all;
9   use IEEE.STD_LOGIC_signed.all;
10
11
12  library work;
13  use work.fltb_par.all;
14
15
16  entity butterfly is
17    generic(
18        wordsize:integer);
19    port(
20        Btf_clk:in std_logic;
21        Btf_ctrl_cnt_in:in std_logic_vector(3 downto 0);
22        Btf_in_0_real:in std_logic_vector(wordsize-1 downto 0);
23        Btf_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
24        Btf_in_1_real:in std_logic_vector(wordsize-1 downto 0);
25        Btf_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
26        Btf_out_0_real:out std_logic_vector(wordsize downto 0);
27        Btf_out_0_imag:out std_logic_vector(wordsize downto 0);
28        Btf_out_1_real:out std_logic_vector(wordsize downto 0);
29        Btf_out_1_imag:out std_logic_vector(wordsize downto 0);
30        Btf_ctrl_cnt_out:out std_logic_vector(3 downto 0));
31  end butterfly;
32
33
34
35  architecture butterfly_arch of butterfly is
36
37  begin
38      process(Btf_clk)
39          begin
40          if Btf_CLK'event and Btf_CLK ='1' then
41              Btf_out_0_real <= (Btf_in_0_real(wordsize-1)&Btf_in_0_real) +
42                 (Btf_in_1_real(wordsize-1)&Btf_in_1_real);
43              Btf_out_1_real <= (Btf_in_0_real(wordsize-1)&Btf_in_0_real) -
44                 (Btf_in_1_real(wordsize-1)&Btf_in_1_real);
45              Btf_out_0_imag <= (Btf_in_0_imag(wordsize-1)&Btf_in_0_imag) +
46                 (Btf_in_1_imag(wordsize-1)&Btf_in_1_imag);
47              Btf_out_1_imag <= (Btf_in_0_imag(wordsize-1)&Btf_in_0_imag) -
48                 (Btf_in_1_imag(wordsize-1)&Btf_in_1_imag);
49              Btf_ctrl_cnt_out <= Btf_ctrl_cnt_in;
50          end if;
51      end process;
52
53  end butterfly_arch;
54
```

```
1  ----------------------------------------------------------------------
2  ----------------------fft_comm.vhd-----------------------------------
3  ----------------------------------------------------------------------
4  --This file describes the commutator, which is used in the pipeline FFT
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14
15 entity fft_comm is
16   generic(
17       wordsize:integer;
18       controlbit:integer);  --0 to 3,
19       --determines which bit of the control counter switches the commutator
20   port(
21       Com_clk:in std_logic;
22       Com_ctrl_cnt_in:in std_logic_vector(3 downto 0);
23       Com_in_0_real:in std_logic_vector(wordsize-1 downto 0);
24       Com_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
25       Com_in_1_real:in std_logic_vector(wordsize-1 downto 0);
26       Com_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
27       Com_out_0_real:out std_logic_vector(wordsize-1 downto 0);
28       Com_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
29       Com_out_1_real:out std_logic_vector(wordsize-1 downto 0);
30       Com_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
31       Com_ctrl_cnt_out:out std_logic_vector(3 downto 0));
32 end fft_comm;
33
34
35
36 architecture fft_comm_arch of fft_comm is
37
38 signal ctrl_bit:std_logic;
39
40 begin
41
42  G0:if controlbit = 0 generate
43         process(Com_ctrl_cnt_in)
44           begin
45           ctrl_bit <= Com_ctrl_cnt_in(0);
46         end process;
47     end generate G0;
48  G1:if controlbit = 1 generate
49         process(Com_ctrl_cnt_in)
50           begin
51           ctrl_bit <= Com_ctrl_cnt_in(1);
52         end process;
53     end generate G1;
54  G2:if controlbit = 2 generate
55         process(Com_ctrl_cnt_in)
56           begin
57           ctrl_bit <= Com_ctrl_cnt_in(2);
```

```
58          end process;
59       end generate G2;
60    G3:if controlbit = 3 generate
61          process(Com_ctrl_cnt_in)
62            begin
63            ctrl_bit <= Com_ctrl_cnt_in(3);
64          end process;
65       end generate G3;
66
67       process(Com_clk)
68          begin
69          if Com_CLK'event and Com_CLK ='1' then
70              if ctrl_bit = '0' then
71                  Com_out_0_real <= Com_in_0_real;
72                  Com_out_0_imag <= Com_in_0_imag;
73                  Com_out_1_real <= Com_in_1_real;
74                  Com_out_1_imag <= Com_in_1_imag;
75              else
76                  Com_out_0_real <= Com_in_1_real;
77                  Com_out_0_imag <= Com_in_1_imag;
78                  Com_out_1_real <= Com_in_0_real;
79                  Com_out_1_imag <= Com_in_0_imag;
80              end if;
81              Com_ctrl_cnt_out <= Com_ctrl_cnt_in;
82          end if;
83       end process;
84
85    end fft_comm_arch;
86
```

```
1  ----------------------------------------------------------------------
2  ----------------------delay.vhd---------------------------------------
3  ----------------------------------------------------------------------
4  --This file describes the delay used in the pipeline FFT
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14
15 entity delay is
16   generic(
17       stage:integer;
18       delayed_path:integer;
19       wordsize:integer
20       );
21
22
23   port(
24       Del_clk:in std_logic;
25       Del_ctrl_cnt_in:in std_logic_vector(3 downto 0);
26       Del_data_in_0_real:in std_logic_vector(wordsize-1 downto 0);
27       Del_data_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
28       Del_data_in_1_real:in std_logic_vector(wordsize-1 downto 0);
29       Del_data_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
30       Del_data_out_0_real:out std_logic_vector(wordsize-1 downto 0);
31       Del_data_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
32       Del_data_out_1_real:out std_logic_vector(wordsize-1 downto 0);
33       Del_data_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
34       Del_ctrl_cnt_out:out std_logic_vector(3 downto 0));
35 end delay;
36
37
38
39 architecture delay_arch of delay is
40
41 type reg1 is array (INTEGER range 0 to stage-1) of
42    std_logic_vector(wordsize-1 downto 0);
43 type reg2 is array (INTEGER range 0 to stage-1) of
44    std_logic_vector(3 downto 0);
45
46 signal data_reg_real,data_reg_imag:reg1;
47 signal ctrl_cnt_reg:reg2;
48
49
50 begin
51
52  G0:if delayed_path = 0 generate   --the top data path and the control
53                                    --counter content will be delayed
54         process(Del_clk)
55
56
57           begin
```

```
58              if Del_CLK'event and Del_CLK ='1' then
59               L1:for i in 0 to stage-2 loop
60                      data_reg_real(i) <= data_reg_real(i+1);
61                      data_reg_imag(i) <= data_reg_imag(i+1);
62                  end loop L1;
63                  data_reg_real(stage-1) <= Del_data_in_0_real;
64                  data_reg_imag(stage-1) <= Del_data_in_0_imag;
65              end if;
66          end process;
67          process(Del_data_in_1_real,Del_data_in_1_imag,data_reg_real,
68             data_reg_imag)
69              begin
70              Del_data_out_0_real <= data_reg_real(0);
71              Del_data_out_0_imag <= data_reg_imag(0);
72              Del_data_out_1_real <= Del_data_in_1_real;
73              Del_data_out_1_imag <= Del_data_in_1_imag;
74          end process;
75
76      process(Del_clk)
77          begin
78              if Del_CLK'event and Del_CLK ='1' then
79               L1:for i in 0 to stage-2 loop
80                      ctrl_cnt_reg(i) <= ctrl_cnt_reg(i+1);
81                  end loop L1;
82                  ctrl_cnt_reg(stage-1) <= Del_ctrl_cnt_in;
83              end if;
84      end process;
85      process(ctrl_cnt_reg)
86          begin
87          Del_ctrl_cnt_out <= ctrl_cnt_reg(0);
88      end process;
89
90  end generate G0;
91
92
93
94
95
96   G1:if delayed_path = 1 generate   --the lower data path will be delayed
97          process(Del_clk)
98              begin
99              if Del_CLK'event and Del_CLK ='1' then
100              L1:for i in 0 to stage-2 loop
101                     data_reg_real(i) <= data_reg_real(i+1);
102                     data_reg_imag(i) <= data_reg_imag(i+1);
103                 end loop L1;
104                data_reg_real(stage-1) <=  Del_data_in_1_real;
105                data_reg_imag(stage-1) <=  Del_data_in_1_imag;
106             end if;
107         end process;
108         process(Del_data_in_0_real,Del_data_in_0_imag,data_reg_real,
109            data_reg_imag)
110             begin
111             Del_data_out_0_real <= Del_data_in_0_real;
112             Del_data_out_0_imag <= Del_data_in_0_imag;
113             Del_data_out_1_real <= data_reg_real(0);
114             Del_data_out_1_imag <= data_reg_imag(0);
```

```
115             end process;
116
117         process(Del_ctrl_cnt_in)
118             begin
119             Del_ctrl_cnt_out <= Del_ctrl_cnt_in;
120         end process;
121
122         end generate G1;
123
124
125     end delay_arch;
126
```

```
1  ----------------------------------------------------------------------------
2  ----------------------mul2val.vhd-------------------------------------------
3  ----------------------------------------------------------------------------
4  --Describes the multiplier of the FFT, which multiplies by 1 or j
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14 entity mul2val is
15   generic(
16       wordsize:integer;
17       controlbit:integer);  --0 to 3, determines which bit of the control
18       --counter selects the factor (1 or j)
19   port(
20       M2_clk:in std_logic;
21       M2_ctrl_cnt_in:in std_logic_vector(3 downto 0);
22       M2_in_0_real:in std_logic_vector(wordsize-1 downto 0);
23       M2_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
24       M2_in_1_real:in std_logic_vector(wordsize-1 downto 0);
25       M2_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
26       M2_out_0_real:out std_logic_vector(wordsize-1 downto 0);
27       M2_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
28       M2_out_1_real:out std_logic_vector(wordsize-1 downto 0);
29       M2_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
30       M2_ctrl_cnt_out:out std_logic_vector(3 downto 0));
31 end mul2val;
32
33
34
35 architecture mul2val_arch of mul2val is
36
37 signal ctrl_bit:std_logic;
38
39 begin
40
41  G0:if controlbit = 0 generate
42        process(M2_ctrl_cnt_in)
43          begin
44          ctrl_bit <= M2_ctrl_cnt_in(0);
45        end process;
46     end generate G0;
47  G1:if controlbit = 1 generate
48        process(M2_ctrl_cnt_in)
49          begin
50          ctrl_bit <= M2_ctrl_cnt_in(1);
51        end process;
52     end generate G1;
53  G2:if controlbit = 2 generate
54        process(M2_ctrl_cnt_in)
55          begin
56          ctrl_bit <= M2_ctrl_cnt_in(2);
57        end process;
```

```
58        end generate G2;
59     G3:if controlbit = 3 generate
60           process(M2_ctrl_cnt_in)
61             begin
62             ctrl_bit <= M2_ctrl_cnt_in(3);
63           end process;
64        end generate G3;
65
66        process(M2_clk)
67           begin
68           if M2_CLK'event and M2_CLK ='1' then
69               if ctrl_bit = '0' then
70                   M2_out_0_real <= M2_in_0_real;
71                   M2_out_0_imag <= M2_in_0_imag;
72                   M2_out_1_real <= M2_in_1_real;
73                   M2_out_1_imag <= M2_in_1_imag;
74               else
75                   M2_out_0_real <= M2_in_0_real;
76                   M2_out_0_imag <= M2_in_0_imag;
77                   M2_out_1_real <= -M2_in_1_imag;
78                   M2_out_1_imag <= M2_in_1_real;
79               end if;
80               M2_ctrl_cnt_out <= M2_ctrl_cnt_in;
81           end if;
82        end process;
83
84    end mul2val_arch;
85
```

```
1  ----------------------------------------------------------------------------
2  ------------------------------mul4val.vhd-----------------------------------
3  ----------------------------------------------------------------------------
4  --Describes the multiplier of the FFT, which multiplies by 1, 1+j, j, -1+j
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14 entity mul4val is
15   generic(
16       ws:integer:=20;  --wordsize
17       ctrl_word_LSB:integer);
18   port(
19       M4_clk:in std_logic;
20       M4_ctrl_cnt_in:in std_logic_vector(3 downto 0);
21       M4_in_0_real:in std_logic_vector(ws-1 downto 0);
22       M4_in_0_imag:in std_logic_vector(ws-1 downto 0);
23       M4_in_1_real:in std_logic_vector(ws-1 downto 0);
24       M4_in_1_imag:in std_logic_vector(ws-1 downto 0);
25       M4_out_0_real:out std_logic_vector(ws-1 downto 0);
26       M4_out_0_imag:out std_logic_vector(ws-1 downto 0);
27       M4_out_1_real:out std_logic_vector(ws-1 downto 0);
28       M4_out_1_imag:out std_logic_vector(ws-1 downto 0);
29       M4_ctrl_cnt_out:out std_logic_vector(3 downto 0));
30 end mul4val;
31
32
33
34 architecture mul4val_arch of mul4val is
35
36 component mul20_8 port (
37       a: IN std_logic_VECTOR(19 downto 0);
38       b: IN std_logic_VECTOR(7 downto 0);
39       c: IN std_logic;
40       ce: IN std_logic;
41       prod: OUT std_logic_VECTOR(27 downto 0));
42 end component;
43
44
45 signal ctrl_word_in,ctrl_word_1:std_logic_vector(1 downto 0);
46 signal ctrl_cnt_stage1,ctrl_cnt_stage2,ctrl_cnt_stage3:
47    std_logic_vector(3 downto 0);
48 signal ctrl_cnt_stage4,ctrl_cnt_stage5:std_logic_vector(3 downto 0);
49
50 signal data_0_real_stage1,data_0_real_stage2,data_0_real_stage3:
51    std_logic_vector(ws-1 downto 0);
52 signal data_0_real_stage4,data_0_real_stage5:std_logic_vector(ws-1 downto
53 0);
54 signal data_0_imag_stage1,data_0_imag_stage2,data_0_imag_stage3:
55    std_logic_vector(ws-1 downto 0);
56 signal data_0_imag_stage4,data_0_imag_stage5:std_logic_vector(ws-1 downto
57 0);
```

```
58
59  signal mult_real_data_in,mult_imag_data_in:std_logic_vector(19 downto 0);
60
61  signal mult_real_fact_in,mult_imag_fact_in:std_logic_vector(7 downto 0);
62
63  signal prod_real,prod_imag:std_logic_vector(27 downto 0);
64
65  signal aaa:std_logic_vector(19 downto 0);
66
67  signal ce:std_logic;
68
69  begin
70
71   G0:if ctrl_word_LSB = 0 generate  --if an 8 pt FFT is being built,
72          --control this multiplier with the lowest two bits
73          process(M4_ctrl_cnt_in,ctrl_cnt_stage1)
74            begin
75            ctrl_word_1 <= ctrl_cnt_stage1(1 downto 0);
76            ctrl_word_in <= M4_ctrl_cnt_in(1 downto 0);
77          end process;
78      end generate G0;
79   G1:if ctrl_word_LSB = 1 generate  --if an 16 pt FFT is being built,
80          --don't use the LSB
81          process(M4_ctrl_cnt_in,ctrl_cnt_stage1)
82            begin
83            ctrl_word_1 <= ctrl_cnt_stage1(2 downto 1);
84            ctrl_word_in <= M4_ctrl_cnt_in(2 downto 1);
85          end process;
86      end generate G1;
87   G2:if ctrl_word_LSB = 2 generate
88          process(M4_ctrl_cnt_in,ctrl_cnt_stage1)
89            begin
90            ctrl_word_1 <= ctrl_cnt_stage1(3 downto 2);
91            ctrl_word_in <= M4_ctrl_cnt_in(3 downto 2);
92          end process;
93      end generate G2;
94
95      process(M4_clk)  --This process delays only unmodified data and
96          --control signals
97          begin
98          if M4_CLK'event and M4_CLK ='1' then
99              ctrl_cnt_stage5 <= ctrl_cnt_stage4;
100             ctrl_cnt_stage4 <= ctrl_cnt_stage3;
101             ctrl_cnt_stage3 <= ctrl_cnt_stage2;
102             ctrl_cnt_stage2 <= ctrl_cnt_stage1;
103             ctrl_cnt_stage1 <= M4_ctrl_cnt_in;
104
105             data_0_real_stage5 <= data_0_real_stage4;
106             data_0_real_stage4 <= data_0_real_stage3;
107             data_0_real_stage3 <= data_0_real_stage2;
108             data_0_real_stage2 <= data_0_real_stage1;
109             data_0_real_stage1 <= M4_in_0_real;
110
111             data_0_imag_stage5 <= data_0_imag_stage4;
112             data_0_imag_stage4 <= data_0_imag_stage3;
113             data_0_imag_stage3 <= data_0_imag_stage2;
114             data_0_imag_stage2 <= data_0_imag_stage1;
```

```
115             data_0_imag_stage1 <= M4_in_0_imag;
116         end if;
117     end process;
118
119     process(M4_clk)
120         begin
121         if M4_CLK'event and M4_CLK ='1' then
122             case ctrl_word_in is
123                 when "00" => mult_real_data_in <= CONV_STD_LOGIC_VECTOR
124                                 (CONV_INTEGER(M4_in_1_real),20);
125                              mult_imag_data_in <= CONV_STD_LOGIC_VECTOR
126                                 (CONV_INTEGER(M4_in_1_imag),20);
127                 when "10" => mult_real_data_in <= CONV_STD_LOGIC_VECTOR
128                                 (CONV_INTEGER(M4_in_1_real),20) +
129                                     CONV_STD_LOGIC_VECTOR((-1)*CONV_INTEGER
130                                        (M4_in_1_imag),20);
131                              mult_imag_data_in <= CONV_STD_LOGIC_VECTOR
132                                 (CONV_INTEGER(M4_in_1_real),20) +
133                                     CONV_STD_LOGIC_VECTOR(CONV_INTEGER
134                                        (M4_in_1_imag),20);
135                 when "01" => mult_real_data_in <= CONV_STD_LOGIC_VECTOR
136                                 ((-1)*CONV_INTEGER(M4_in_1_imag),20);
137                              mult_imag_data_in <= CONV_STD_LOGIC_VECTOR
138                                 (CONV_INTEGER(M4_in_1_real),20);
139                 when "11" => mult_real_data_in <= CONV_STD_LOGIC_VECTOR
140                                 ((-1)*CONV_INTEGER(M4_in_1_real),20) +
141                                    CONV_STD_LOGIC_VECTOR((-1)*CONV_INTEGER
142                                       (M4_in_1_imag),20);
143                              mult_imag_data_in <= CONV_STD_LOGIC_VECTOR
144                                 (CONV_INTEGER(M4_in_1_real),20) +
145                                    CONV_STD_LOGIC_VECTOR((-1)*CONV_INTEGER
146                                       (M4_in_1_imag),20);
147
148                 when others =>
149             end case;
150        end if;
151     end process;
152
153
154
155     process(M4_clk)
156         begin
157         if M4_CLK'event and M4_CLK ='1' then
158             case ctrl_word_in is
159                 when "00" => mult_real_fact_in
160                                 <= CONV_STD_LOGIC_VECTOR(127,8);
161                              mult_imag_fact_in
162                                 <= CONV_STD_LOGIC_VECTOR(127,8);
163                 when "01" => mult_real_fact_in
164                                 <= CONV_STD_LOGIC_VECTOR(127,8);
165                              mult_imag_fact_in
166                                 <= CONV_STD_LOGIC_VECTOR(127,8);
167                 when "10" => mult_real_fact_in
168                                 <= CONV_STD_LOGIC_VECTOR(91,8);
169                                 --91=128*0.707
170                              mult_imag_fact_in
171                                 <= CONV_STD_LOGIC_VECTOR(91,8);
```

```
172                  when "11" => mult_real_fact_in
173                                  <= CONV_STD_LOGIC_VECTOR(91,8);
174                               mult_imag_fact_in
175                                  <= CONV_STD_LOGIC_VECTOR(91,8);
176
177                  when others =>
178              end case;
179           end if;
180        end process;
181
182        process(M4_clk)
183           begin
184              ce <= '1';
185        end process;
186
187        mul20_8_real : mul20_8 port map (
188              a => mult_real_data_in,
189              b => mult_real_fact_in,
190              c => M4_clk,
191              ce => ce,
192              prod => prod_real);
193
194        mul20_8_imag : mul20_8 port map (
195              a => mult_imag_data_in,
196              b => mult_imag_fact_in,
197              c => M4_clk,
198              ce => ce,
199              prod => prod_imag);
200
201
202        process(ctrl_cnt_stage5,data_0_real_stage5,data_0_imag_stage5,
203                                                   prod_real,prod_imag)
204            begin
205            M4_out_0_real <= data_0_real_stage5;
206            M4_out_0_imag <= data_0_imag_stage5;
207
208            M4_out_1_real <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
209               (prod_real(26 downto 7)),ws);
210            M4_out_1_imag <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
211               (prod_imag(26 downto 7)),ws);
212            M4_ctrl_cnt_out <= ctrl_cnt_stage5;
213        end process;
214
215
216  end mul4val_arch;
217
```

```
1  ----------------------------------------------------------------------------
2  -------------------------mul8val.vhd--------------------------------------
3  ----------------------------------------------------------------------------
4  --Describes the multiplier of the FFT/IFFT, which multiplies by 8 values
5  --spaced on the upper unit circle.
6
7  library IEEE;
8  use IEEE.STD_LOGIC_1164.all;
9  use IEEE.STD_LOGIC_arith.all;
10 use IEEE.STD_LOGIC_signed.all;
11
12 library work;
13 use work.fltb_par.all;
14
15 entity mul8val is
16   generic(
17       ws:integer:=20;  --wordsize
18       ctrl_word_LSB:integer);
19   port(
20       M8_clk:in std_logic;
21       M8_ctrl_cnt_in:in std_logic_vector(3 downto 0);
22       M8_in_0_real:in std_logic_vector(ws-1 downto 0);
23       M8_in_0_imag:in std_logic_vector(ws-1 downto 0);
24       M8_in_1_real:in std_logic_vector(ws-1 downto 0);
25       M8_in_1_imag:in std_logic_vector(ws-1 downto 0);
26       M8_out_0_real:out std_logic_vector(ws-1 downto 0);
27       M8_out_0_imag:out std_logic_vector(ws-1 downto 0);
28       M8_out_1_real:out std_logic_vector(ws-1 downto 0);
29       M8_out_1_imag:out std_logic_vector(ws-1 downto 0);
30       M8_ctrl_cnt_out:out std_logic_vector(3 downto 0));
31 end mul8val;
32
33
34
35 architecture mul8val_arch of mul8val is
36
37 component mul20_8 port (
38       a: IN std_logic_VECTOR(19 downto 0);
39       b: IN std_logic_VECTOR(7 downto 0);
40       c: IN std_logic;
41       ce: IN std_logic;
42       prod: OUT std_logic_VECTOR(27 downto 0));
43 end component;
44
45
46 signal ctrl_word_in,ctrl_word_1:std_logic_vector(2 downto 0);
47 signal ctrl_cnt_stage1,ctrl_cnt_stage2,ctrl_cnt_stage3:
48    std_logic_vector(3 downto 0);
49 signal ctrl_cnt_stage4,ctrl_cnt_stage5,ctrl_cnt_stage6:
50    std_logic_vector(3 downto 0);
51
52 signal data_0_real_stage1,data_0_real_stage2,data_0_real_stage3:
53    std_logic_vector(ws-1 downto 0);
54 signal data_0_real_stage4,data_0_real_stage5,data_0_real_stage6:
55    std_logic_vector(ws-1 downto 0);
56 signal data_0_imag_stage1,data_0_imag_stage2,data_0_imag_stage3:
57    std_logic_vector(ws-1 downto 0);
```

```
58  signal data_0_imag_stage4,data_0_imag_stage5,data_0_imag_stage6:
59     std_logic_vector(ws-1 downto 0);
60
61  signal mult1_data_in,mult2_data_in,mult3_data_in:
62     std_logic_vector(19 downto 0);
63
64  signal mult1_fact_in,mult2_fact_in,mult3_fact_in:
65     std_logic_vector(7 downto 0);
66
67  signal prod1,prod2,prod3:std_logic_vector(27 downto 0);
68
69  signal ce:std_logic;
70
71  begin
72
73   G0:if ctrl_word_LSB = 0 generate  --if a 16 pt FFT is being built,
74          --control this multiplier with the lowest three bits
75          process(M8_ctrl_cnt_in,ctrl_cnt_stage1)
76            begin
77            ctrl_word_1 <= ctrl_cnt_stage1(2 downto 0);
78            ctrl_word_in <= M8_ctrl_cnt_in(2 downto 0);
79          end process;
80      end generate G0;
81   G1:if ctrl_word_LSB = 1 generate  --if a 32 pt FFT is being built,
82          --don't use the LSB
83          process(M8_ctrl_cnt_in,ctrl_cnt_stage1)
84            begin
85            ctrl_word_1 <= ctrl_cnt_stage1(3 downto 1);
86            ctrl_word_in <= M8_ctrl_cnt_in(3 downto 1);
87          end process;
88      end generate G1;
89
90
91      process(M8_clk)  --This process delays only unmodified data
92          --and control signals
93          begin
94          if M8_CLK'event and M8_CLK ='1' then
95              ctrl_cnt_stage6 <= ctrl_cnt_stage5;
96              ctrl_cnt_stage5 <= ctrl_cnt_stage4;
97              ctrl_cnt_stage4 <= ctrl_cnt_stage3;
98              ctrl_cnt_stage3 <= ctrl_cnt_stage2;
99              ctrl_cnt_stage2 <= ctrl_cnt_stage1;
100             ctrl_cnt_stage1 <= M8_ctrl_cnt_in;
101
102             data_0_real_stage6 <= data_0_real_stage5;
103             data_0_real_stage5 <= data_0_real_stage4;
104             data_0_real_stage4 <= data_0_real_stage3;
105             data_0_real_stage3 <= data_0_real_stage2;
106             data_0_real_stage2 <= data_0_real_stage1;
107             data_0_real_stage1 <= M8_in_0_real;
108
109             data_0_imag_stage6 <= data_0_imag_stage5;
110             data_0_imag_stage5 <= data_0_imag_stage4;
111             data_0_imag_stage4 <= data_0_imag_stage3;
112             data_0_imag_stage3 <= data_0_imag_stage2;
113             data_0_imag_stage2 <= data_0_imag_stage1;
114             data_0_imag_stage1 <= M8_in_0_imag;
```

```
115         end if;
116       end process;
117
118
119       process(M8_clk)
120           begin
121           if M8_CLK'event and M8_CLK ='1' then
122               mult1_data_in <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
123                  (M8_in_1_real)+CONV_INTEGER(M8_in_1_imag),20);
124               mult2_data_in <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
125                  (M8_in_1_real),20);
126               mult3_data_in <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
127                  (M8_in_1_real)-CONV_INTEGER(M8_in_1_imag),20);
128          end if;
129       end process;
130
131
132       process(M8_clk)
133           begin
134           if M8_CLK'event and M8_CLK ='1' then
135               case ctrl_word_in is
136                   when "000" => mult1_fact_in <= CONV_STD_LOGIC_VECTOR(0,8);
137                                 mult2_fact_in <= CONV_STD_LOGIC_VECTOR(63,8);
138                                 mult3_fact_in <= CONV_STD_LOGIC_VECTOR(63,8);
139                   when "001" => mult1_fact_in <= CONV_STD_LOGIC_VECTOR(63,8);
140                                 mult2_fact_in <= CONV_STD_LOGIC_VECTOR(63,8);
141                                 mult3_fact_in <= CONV_STD_LOGIC_VECTOR(0,8);
142                   when "010" => mult1_fact_in <= CONV_STD_LOGIC_VECTOR(45,8);
143                                 mult2_fact_in <= CONV_STD_LOGIC_VECTOR(91,8);
144                                 mult3_fact_in <= CONV_STD_LOGIC_VECTOR(45,8);
145                   when "011" => mult1_fact_in <= CONV_STD_LOGIC_VECTOR(45,8);
146                                 mult2_fact_in <= CONV_STD_LOGIC_VECTOR(0,8);
147                                 mult3_fact_in <= CONV_STD_LOGIC_VECTOR(-45,8);
148                   when "100" => mult1_fact_in <= CONV_STD_LOGIC_VECTOR(32,8);
149                                 mult2_fact_in <= CONV_STD_LOGIC_VECTOR(87,8);
150                                 mult3_fact_in <= CONV_STD_LOGIC_VECTOR(55,8);
151                   when "101" => mult1_fact_in <= CONV_STD_LOGIC_VECTOR(55,8);
152                                 mult2_fact_in <= CONV_STD_LOGIC_VECTOR(23,8);
153                                 mult3_fact_in <= CONV_STD_LOGIC_VECTOR(-32,8);
154                   when "110" => mult1_fact_in <= CONV_STD_LOGIC_VECTOR(55,8);
155                                 mult2_fact_in <= CONV_STD_LOGIC_VECTOR(87,8);
156                                 mult3_fact_in <= CONV_STD_LOGIC_VECTOR(32,8);
157                   when "111" => mult1_fact_in <= CONV_STD_LOGIC_VECTOR(32,8);
158                                 mult2_fact_in <= CONV_STD_LOGIC_VECTOR(-23,8);
159                                 mult3_fact_in <= CONV_STD_LOGIC_VECTOR(-55,8);
160
161                   when others =>
162              end case;
163          end if;
164       end process;
165
166       process(M8_clk)
167          begin
168             ce <= '1';
169       end process;
170
171
```

```
172       mul20_8_1 : mul20_8 port map (
173             a => mult1_data_in,
174             b => mult1_fact_in,
175             c => M8_clk,
176             ce => ce,
177             prod => prod1);
178
179       mul20_8_2 : mul20_8 port map (
180             a => mult2_data_in,
181             b => mult2_fact_in,
182             c => M8_clk,
183             ce => ce,
184             prod => prod2);
185
186       mul20_8_3: mul20_8 port map (
187             a => mult3_data_in,
188             b => mult3_fact_in,
189             c => M8_clk,
190             ce => ce,
191             prod => prod3);
192
193       process(M8_clk)
194           begin
195           if M8_CLK'event and M8_CLK ='1' then
196           M8_out_1_real <= CONV_STD_LOGIC_VECTOR(((-1)*CONV_INTEGER(prod1)
197              +CONV_INTEGER(prod2))/64,ws);
198           M8_out_1_imag <= CONV_STD_LOGIC_VECTOR((CONV_INTEGER(prod2)
199              -CONV_INTEGER(prod3))/64,ws);
200          end if;
201       end process;
202
203
204       process(ctrl_cnt_stage6,data_0_real_stage6,data_0_imag_stage6,prod1,
205                                                                  prod2,prod3)
206           begin
207           M8_out_0_real <= data_0_real_stage6;
208           M8_out_0_imag <= data_0_imag_stage6;
209           M8_ctrl_cnt_out <= ctrl_cnt_stage6;
210       end process;
211
212
213   end mul8val_arch;
214
```

```
1  --------------------------------------------------------------------------
2  ------------------------outp_sr.vhd---------------------------------------
3  --------------------------------------------------------------------------
4  --This file describes the output shift register of the pipeline FFT
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14 entity fft_out_reg is
15   generic(
16       wordsize:integer);
17   port(
18       SR_clk:in std_logic;
19       SR_ctrl_cnt_in:in std_logic_vector(3 downto 0);
20       SR_data_in_0_real:in std_logic_vector(wordsize-1 downto 0);
21       SR_data_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
22       SR_data_in_1_real:in std_logic_vector(wordsize-1 downto 0);
23       SR_data_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
24       SR_rdy_out:out std_logic;
25       SR_output_real:out fft_out_data;         --array
26       SR_output_imag:out fft_out_data);         --array
27 end fft_out_reg;
28
29
30
31 architecture fft_out_reg_arch of fft_out_reg is
32
33 signal reg_real,reg_imag:fft_out_data;     --array
34
35 begin
36
37 process(SR_clk)
38
39     begin
40
41     if SR_CLK'event and SR_CLK ='1' then
42         if SR_ctrl_cnt_in = CONV_STD_LOGIC_VECTOR(nmb_of_FFT_pt/2-1,4) then
43             SR_rdy_out <= '1';
44         else
45             SR_rdy_out <= '0';
46         end if;
47      if nmb_of_FFT_pt > 2 then
48         L1:for i in 0 to abs(nmb_of_FFT_pt/2-2) loop --abs to get
49                                       --it compiled
50             reg_real(i) <= reg_real(i+1);
51             reg_imag(i) <= reg_imag(i+1);
52            end loop L1;
53      end if;
54         reg_real(nmb_of_FFT_pt/2-1) <= SR_data_in_0_real;
55         reg_imag(nmb_of_FFT_pt/2-1) <= SR_data_in_0_imag;
56      L2:for i in nmb_of_FFT_pt/2 to nmb_of_FFT_pt-2 loop
57             reg_real(i) <= reg_real(i+1);
```

```
58              reg_imag(i) <= reg_imag(i+1);
59          end loop L2;
60          reg_real(nmb_of_FFT_pt-1) <= SR_data_in_1_real;
61          reg_imag(nmb_of_FFT_pt-1) <= SR_data_in_1_imag;
62
63      end if;
64
65  end process;
66
67  process(reg_real,reg_imag)   --reorder output sequence
68      constant rev_index:int_array:=
69  --                  (0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,        --for 2 pt FFT
70  --                   0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3,        --for 4 pt FFT
71  --                   0,2,1,3,4,6,5,7,0,2,1,3,4,6,5,7,        --for 8 pt FFT
72  --                   0,4,2,6,1,5,3,7,8,12,10,14,9,13,11,15); --for 16 pt FFT
73
74                       (0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,        --for 2 pt FFT
75                        0,3,2,1,0,3,2,1,0,3,2,1,0,3,2,1,        --for 4 pt FFT
76                        0,7,5,6,4,3,1,2,0,7,5,6,4,3,1,2,        --for 8 pt FFT
77                        0,15,11,13,9,14,10,12,8,7,3,5,1,6,2,4); --for 16 pt FFT
78
79      --Why is MSB not involved in the bit reversal scheme ?
80      --Look book Oppenheim, p.597, fig. 9.14: The output of the upper
81      --butterfly branch has always index with MSB cleared, the output of the
82      --lower branch has always the MSB set.
83
84      begin
85  L10:for i in 0 to nmb_of_FFT_pt-1 loop
86          SR_output_real(i) <= reg_real(rev_index((log(nmb_of_FFT_pt)-
87  1)*16+i));
88          SR_output_imag(i) <= reg_imag(rev_index((log(nmb_of_FFT_pt)-
89  1)*16+i));
90
91      end loop L10;
92
93  end process;
94
95  end fft_out_reg_arch;
96
97
```

```
1  ----------------------------------------------------------------------------
2  ----------------------sy_n.vhd-------------------------------------------
3  ----------------------------------------------------------------------------
4  --This file describes the synthesis filter bank
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14 entity Sy_n is
15   port(
16        clk :in STD_LOGIC;
17        reset: in STD_LOGIC;
18        Sy_rdy_in:in std_logic;
19        Sy_data_in_real:in Sy_in_data;
20        Sy_data_in_imag:in Sy_in_data;
21        Sy_data_out:out STD_LOGIC_VECTOR (Sy_out_ws-1 downto 0);
22        Wr_not:in Wr_not_type;
23        Adr_Sy:in Adr_Sy_type;
24        downsampl_del:in STD_LOGIC);
25 end Sy_n;
26
27
28
29 architecture Sy_n of Sy_n is
30
31 --signal rdy_in:STD_LOGIC;
32 signal ifft_data_in_real:IFFT_in_data;
33 signal ifft_data_in_imag:IFFT_in_data;
34 signal ifft_data_out_real,ifft_data_out_imag:IFFT_out_data;
35 signal ifft_rdy_out:std_logic;
36 signal dwsby2_data_in:dwsby2_data;
37 signal Com_Sy_data_in:dwsby2_data;
38 signal Com_Sy_data_out:STD_LOGIC_VECTOR (Com_Sy_ws-1 downto 0);
39 signal Sy_fil_data_in:Sy_fil_in_data;
40 signal Sy_fil_data_out:Sy_fil_out_data;
41 signal Sy_fil_rdy_out:std_logic;
42 --signal clk2:clk2_type;
43 signal clk_4sb_0,clk_4sb_1,clk_4sb_2:std_logic;
44 signal clk_8sb_0,clk_8sb_1,clk_8sb_2,clk_8sb_3,clk_8sb_4:std_logic;
45 signal clk_16sb_0,clk_16sb_1,clk_16sb_2,clk_16sb_3,clk_16sb_4:std_logic;
46 signal clk_16sb_5,clk_16sb_6,clk_16sb_7,clk_16sb_8:std_logic;
47
48 signal Sy_in_reg_real:Sy_in_data;
49 signal Sy_in_reg_imag:Sy_in_data;
50
51
52    component ifft port(
53       clk:in std_logic;
54       rdy_in:in std_logic;
55       ifft_data_in_real:in IFFT_in_data;
56       ifft_data_in_imag:in IFFT_in_data;
57       ifft_data_out_real:out ifft_out_data;
```

```
58          ifft_data_out_imag:out ifft_out_data;
59          ifft_rdy_out:out std_logic);
60       end component;
61
62       component Sy_fil port(
63          Clk:in STD_LOGIC;
64          Reset:in STD_LOGIC;
65          Sy_fil_data_in:in Sy_fil_in_data;
66          Sy_fil_data_out:out Sy_fil_out_data;
67          Sy_fil_rdy_out:out std_logic);
68       end component;
69
70       component Sy_fil8 port(
71          Clk:in STD_LOGIC;
72          Reset:in STD_LOGIC;
73          Sy_fil_data_in:in Sy_fil_in_data;
74          Sy_fil_data_out:out Sy_fil_out_data;
75          Sy_fil_rdy_out:out STD_LOGIC);
76       end component;
77
78       component Sy_fil16 port(
79          Clk:in STD_LOGIC;
80          Reset:in STD_LOGIC;
81          Sy_fil_data_in:in Sy_fil_in_data;
82          Sy_fil_data_out:out Sy_fil_out_data;
83          Sy_fil_rdy_out:out STD_LOGIC);
84       end component;
85
86
87       component dwsby2_n port(
88          Clk:in STD_LOGIC;
89          Reset:in STD_LOGIC;
90          dwsby2_data_in:in dwsby2_data;
91          dwsby2_data_out:out dwsby2_data;
92          downsampl_del:in STD_LOGIC);
93       end component;
94
95       component com_sy_n port(
96          Clk:in STD_LOGIC;
97          Reset:in STD_LOGIC;
98          Com_Sy_data_in:in dwsby2_data;
99          Com_Sy_data_out:out STD_LOGIC_VECTOR (Com_Sy_ws-1 downto 0) );
100      end component;
101
102
103      begin
104
105   G4:if nmb_of_subband = 4 generate
106         process(Wr_not,Adr_Sy)
107            begin
108               clk_4sb_0 <= NOT ((NOT (Wr_not(0))) AND Adr_Sy(0));
109               clk_4sb_1 <= NOT ((NOT (Wr_not(1))) AND Adr_Sy(1));
110               clk_4sb_2 <= NOT ((NOT (Wr_not(2))) AND Adr_Sy(2));
111         end process;
112
113         process(clk_4sb_0) --input register
114            begin
```

```
115             if (clk_4sb_0'event and clk_4sb_0 = '1') then
116                Sy_in_reg_real(0) <= Sy_data_in_real(0);
117                Sy_in_reg_imag(0) <= Sy_data_in_imag(0);
118             end if;
119       end process;
120
121       process(clk_4sb_1) --input register
122          begin
123             if (clk_4sb_1'event and clk_4sb_1 = '1') then
124                Sy_in_reg_real(1) <= Sy_data_in_real(1);
125                Sy_in_reg_imag(1) <= Sy_data_in_imag(1);
126             end if;
127       end process;
128
129       process(clk_4sb_2) --input register
130          begin
131             if (clk_4sb_2'event and clk_4sb_2 = '1') then
132                Sy_in_reg_real(2) <= Sy_data_in_real(2);
133                Sy_in_reg_imag(2) <= Sy_data_in_imag(2);
134             end if;
135       end process;
136    end generate;
137
138
139 G8:if nmb_of_subband = 8 generate
140       process(Wr_not,Adr_Sy)
141          variable index:integer;
142          begin
143             clk_8sb_0 <= NOT ((NOT (Wr_not(0))) AND Adr_Sy(0));
144             clk_8sb_1 <= NOT ((NOT (Wr_not(1))) AND Adr_Sy(1));
145             clk_8sb_2 <= NOT ((NOT (Wr_not(2))) AND Adr_Sy(2));
146             index := CONV_INTEGER("0011" and CONV_STD_LOGIC_VECTOR
147                (nmb_of_subband-1,4));
148             clk_8sb_3 <= NOT ((NOT (Wr_not(index))) AND Adr_Sy(index));
149             index := CONV_INTEGER("0100" and CONV_STD_LOGIC_VECTOR
150                (nmb_of_subband-1,4));
151             clk_8sb_4 <= NOT ((NOT (Wr_not(index))) AND Adr_Sy(index));
152       end process;
153
154       process(clk_8sb_0) --input register
155          begin
156             if (clk_8sb_0'event and clk_8sb_0 = '1') then
157                Sy_in_reg_real(0) <= Sy_data_in_real(0);
158                Sy_in_reg_imag(0) <= Sy_data_in_imag(0);
159             end if;
160       end process;
161
162       process(clk_8sb_1) --input register
163          begin
164             if (clk_8sb_1'event and clk_8sb_1 = '1') then
165                Sy_in_reg_real(1) <= Sy_data_in_real(1);
166                Sy_in_reg_imag(1) <= Sy_data_in_imag(1);
167             end if;
168       end process;
169
170       process(clk_8sb_2) --input register
171          begin
```

```
172             if (clk_8sb_2'event and clk_8sb_2 = '1') then
173                Sy_in_reg_real(2) <= Sy_data_in_real(2);
174                Sy_in_reg_imag(2) <= Sy_data_in_imag(2);
175             end if;
176       end process;
177
178       process(clk_8sb_3) --input register
179          variable index:integer;
180          begin
181             index := CONV_INTEGER("0011" and CONV_STD_LOGIC_VECTOR
182                (nmb_of_subband-1,4));
183             if (clk_8sb_3'event and clk_8sb_3 = '1') then
184                Sy_in_reg_real(index) <= Sy_data_in_real(index);
185                Sy_in_reg_imag(index) <= Sy_data_in_imag(index);
186             end if;
187       end process;
188
189       process(clk_8sb_4) --input register
190          variable index:integer;
191          begin
192             index := CONV_INTEGER("0100" and CONV_STD_LOGIC_VECTOR
193                (nmb_of_subband-1,4));
194             if (clk_8sb_4'event and clk_8sb_4 = '1') then
195                Sy_in_reg_real(index) <= Sy_data_in_real(index);
196                Sy_in_reg_imag(index) <= Sy_data_in_imag(index);
197             end if;
198       end process;
199
200    end generate;
201
202
203
204    process(Sy_in_reg_real,Sy_in_reg_imag)
205       --Use the Hermitian symmetry of the FFT to determine the inputs
206       --of the IFFT for N/2 < k < N.
207       --The conjugate complex of the FFT is fed into the IFFT, because
208       --X = W*x, x = CT{W}*x = C{W}*x = C{W*C{x}}    C=conjugate,
209       --                                            T=transpose
210       begin
211          L1:for i in 0 to nmb_of_subband/2 loop
212             ifft_data_in_real(i) <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
213                (Sy_in_reg_real(i)),ifft_in_ws);
214             ifft_data_in_imag(i) <= CONV_STD_LOGIC_VECTOR((-1)*CONV_INTEGER
215                (Sy_in_reg_imag(i)),ifft_in_ws);
216          end loop L1;
217          L2:for i in nmb_of_subband/2+1 to nmb_of_subband-1 loop
218             ifft_data_in_real(i) <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
219                (Sy_in_reg_real(nmb_of_subband-i)),ifft_in_ws);
220             ifft_data_in_imag(i) <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
221                (Sy_in_reg_imag(nmb_of_subband-i)),ifft_in_ws);
222          end loop L2;
223
224    end process;
225
226
227
228    ifft_1:ifft port map(
```

```
229         clk => clk,
230         rdy_in => Sy_rdy_in,
231         ifft_data_in_real => ifft_data_in_real,
232         ifft_data_in_imag => ifft_data_in_imag,
233         ifft_data_out_real => ifft_data_out_real,
234         ifft_data_out_imag => ifft_data_out_imag,
235         ifft_rdy_out => ifft_rdy_out);
236
237      process(clk)  --type conversion and latch
238         begin
239            if CLK'event and CLK ='1' then
240               if(ifft_rdy_out = '1') then
241               L1:for i in 0 to nmb_of_FFT_pt-1 loop
242                  sy_fil_data_in(i) <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
243                     (ifft_data_out_real(i)),Sy_filt_in_ws);
244                  end loop L1;
245               end if;
246            end if;
247      end process;
248
249   G1:if Nmb_of_Subband = 4 generate
250      sy_fil_1:sy_fil port map(
251         Clk => clk,
252         Reset => reset,
253         Sy_fil_data_in => Sy_fil_data_in,
254         Sy_fil_data_out => Sy_fil_data_out,
255         Sy_fil_rdy_out => sy_fil_rdy_out);
256      end generate G1;
257
258   G2:if Nmb_of_Subband = 8 generate
259      sy_fil_1:sy_fil8 port map(
260         Clk => clk,
261         Reset => reset,
262         Sy_fil_data_in => Sy_fil_data_in,
263         Sy_fil_data_out => Sy_fil_data_out,
264         Sy_fil_rdy_out => sy_fil_rdy_out);
265      end generate G2;
266
267   G3:if Nmb_of_Subband = 16 generate
268      sy_fil_1:sy_fil16 port map(
269         Clk => clk,
270         Reset => reset,
271         Sy_fil_data_in => Sy_fil_data_in,
272         Sy_fil_data_out => Sy_fil_data_out,
273         Sy_fil_rdy_out => sy_fil_rdy_out);
274      end generate G3;
275
276
277      process(clk)
278         begin              --type conversion and latch
279            if CLK'event and CLK ='1' then
280               if(Sy_fil_rdy_out = '1') then
281               L1:for i in 0 to nmb_of_FFT_pt-1 loop
282                  dwsby2_data_in(i) <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
283                     (Sy_fil_data_out(i)),dwsby2_ws);
284                  end loop L1;
285               end if;
```

```
286             end if;
287       end process;
288
289       dwsby2_n_1:dwsby2_n port map(
290          Clk => clk,
291          Reset => reset,
292          dwsby2_data_in => dwsby2_data_in,
293          dwsby2_data_out => Com_Sy_data_in,
294          downsampl_del => downsampl_del);
295
296       com_sy_n_1:com_sy_n port map(
297          clk => clk,
298          Reset => reset,
299          Com_Sy_data_in => Com_Sy_data_in,
300          Com_Sy_data_out => Com_Sy_data_out);
301
302       process(Com_Sy_data_out)  --only type conversion
303          begin
304             Sy_data_out <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER
305                (Com_Sy_data_out),Sy_out_ws);
306       end process;
307
308   end Sy_n;
```

```
1  ----------------------------------------------------------------------
2  ---------------------------------ifft.vhd------------------------------
3  ----------------------------------------------------------------------
4  --Describes the IFFT, used in the synthesis filter bank
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library work;
12 use work.fltb_par.all;
13
14
15 entity ifft is
16   port(
17       clk:in std_logic;
18       rdy_in:in std_logic;
19       ifft_data_in_real:in ifft_in_data;
20       ifft_data_in_imag:in ifft_in_data;
21       ifft_data_out_real:out ifft_out_data;
22       ifft_data_out_imag:out ifft_out_data;
23       ifft_rdy_out:out std_logic);
24 end ifft;
25
26
27
28 architecture ifft_arch of ifft is
29
30 component ifft_inp_reg
31     generic(
32         wordsize:integer);
33     port(
34         SR_clk:in std_logic;
35         SR_rdy_in:in std_logic;
36         SR_input_real:in ifft_in_data;          --array
37         SR_input_imag:in ifft_in_data;          --array
38         SR_data_out_0_real:out std_logic_vector(wordsize-1 downto 0);
39         SR_data_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
40         SR_data_out_1_real:out std_logic_vector(wordsize-1 downto 0);
41         SR_data_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
42         SR_ctrl_cnt_out:out std_logic_vector(3 downto 0));
43 end component;
44
45 component butterfly
46   generic(
47       wordsize:integer);
48   port(
49       Btf_clk:in std_logic;
50       Btf_ctrl_cnt_in:in std_logic_vector(3 downto 0);
51       Btf_in_0_real:in std_logic_vector(wordsize-1 downto 0);
52       Btf_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
53       Btf_in_1_real:in std_logic_vector(wordsize-1 downto 0);
54       Btf_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
55       Btf_out_0_real:out std_logic_vector(wordsize downto 0);
56       Btf_out_0_imag:out std_logic_vector(wordsize downto 0);
57       Btf_out_1_real:out std_logic_vector(wordsize downto 0);
```

```
58          Btf_out_1_imag:out std_logic_vector(wordsize downto 0);
59          Btf_ctrl_cnt_out:out std_logic_vector(3 downto 0));
60    end component;
61
62    component delay
63      generic(
64          stage:integer;
65          delayed_path:integer;
66          wordsize:integer);
67      port(
68          Del_clk:in std_logic;
69          Del_ctrl_cnt_in:in std_logic_vector(3 downto 0);
70          Del_data_in_0_real:in std_logic_vector(wordsize-1 downto 0);
71          Del_data_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
72          Del_data_in_1_real:in std_logic_vector(wordsize-1 downto 0);
73          Del_data_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
74          Del_data_out_0_real:out std_logic_vector(wordsize-1 downto 0);
75          Del_data_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
76          Del_data_out_1_real:out std_logic_vector(wordsize-1 downto 0);
77          Del_data_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
78          Del_ctrl_cnt_out:out std_logic_vector(3 downto 0));
79    end component;
80
81    component fft_comm
82      generic(
83          wordsize:integer;
84          controlbit:integer);  --0 to 3, determines which bit of the control
85          --counter switches the commutator
86      port(
87          Com_clk:in std_logic;
88          Com_ctrl_cnt_in:in std_logic_vector(3 downto 0);
89          Com_in_0_real:in std_logic_vector(wordsize-1 downto 0);
90          Com_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
91          Com_in_1_real:in std_logic_vector(wordsize-1 downto 0);
92          Com_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
93          Com_out_0_real:out std_logic_vector(wordsize-1 downto 0);
94          Com_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
95          Com_out_1_real:out std_logic_vector(wordsize-1 downto 0);
96          Com_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
97          Com_ctrl_cnt_out:out std_logic_vector(3 downto 0));
98    end component;
99
100   component mul2val
101     generic(
102         wordsize:integer;
103         controlbit:integer);  --0 to 3, determines which bit of the control
104         --counter selects the factor (1 or j)
105     port(
106         M2_clk:in std_logic;
107         M2_ctrl_cnt_in:in std_logic_vector(3 downto 0);
108         M2_in_0_real:in std_logic_vector(wordsize-1 downto 0);
109         M2_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
110         M2_in_1_real:in std_logic_vector(wordsize-1 downto 0);
111         M2_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
112         M2_out_0_real:out std_logic_vector(wordsize-1 downto 0);
113         M2_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
114         M2_out_1_real:out std_logic_vector(wordsize-1 downto 0);
```

```
115         M2_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
116         M2_ctrl_cnt_out:out std_logic_vector(3 downto 0));
117   end component;
118
119   component mul4val
120     generic(
121         ws:integer;  --wordsize
122         ctrl_word_LSB:integer);
123     port(
124         M4_clk:in std_logic;
125         M4_ctrl_cnt_in:in std_logic_vector(3 downto 0);
126         M4_in_0_real:in std_logic_vector(ws-1 downto 0);
127         M4_in_0_imag:in std_logic_vector(ws-1 downto 0);
128         M4_in_1_real:in std_logic_vector(ws-1 downto 0);
129         M4_in_1_imag:in std_logic_vector(ws-1 downto 0);
130         M4_out_0_real:out std_logic_vector(ws-1 downto 0);
131         M4_out_0_imag:out std_logic_vector(ws-1 downto 0);
132         M4_out_1_real:out std_logic_vector(ws-1 downto 0);
133         M4_out_1_imag:out std_logic_vector(ws-1 downto 0);
134         M4_ctrl_cnt_out:out std_logic_vector(3 downto 0));
135   end component;
136
137   component mul8val
138     generic(
139         ws:integer;  --wordsize
140         ctrl_word_LSB:integer);
141     port(
142         M8_clk:in std_logic;
143         M8_ctrl_cnt_in:in std_logic_vector(3 downto 0);
144         M8_in_0_real:in std_logic_vector(ws-1 downto 0);
145         M8_in_0_imag:in std_logic_vector(ws-1 downto 0);
146         M8_in_1_real:in std_logic_vector(ws-1 downto 0);
147         M8_in_1_imag:in std_logic_vector(ws-1 downto 0);
148         M8_out_0_real:out std_logic_vector(ws-1 downto 0);
149         M8_out_0_imag:out std_logic_vector(ws-1 downto 0);
150         M8_out_1_real:out std_logic_vector(ws-1 downto 0);
151         M8_out_1_imag:out std_logic_vector(ws-1 downto 0);
152         M8_ctrl_cnt_out:out std_logic_vector(3 downto 0));
153   end component;
154
155   component ifft_out_reg
156     generic(
157         wordsize:integer);
158     port(
159         SR_clk:in std_logic;
160         SR_ctrl_cnt_in:in std_logic_vector(3 downto 0);
161         SR_data_in_0_real:in std_logic_vector(wordsize-1 downto 0);
162         SR_data_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
163         SR_data_in_1_real:in std_logic_vector(wordsize-1 downto 0);
164         SR_data_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
165         SR_rdy_out:out std_logic;
166         SR_output_real:out ifft_out_data;          --array
167         SR_output_imag:out ifft_out_data);          --array
168   end component;
169
170   ----------SIGNALS FOR STAGE 0-----------------------------------------------
171   ----------input shift register signals
```

```
172  signal inp_SR_out_0_r,inp_SR_out_0_i,inp_SR_out_1_r,inp_SR_out_1_i:
173     std_logic_vector(ifft_in_ws-1 downto 0);
174  signal inp_SR_ctrl_cnt_out:std_logic_vector(3 downto 0);
175  ----------signals from butterfly #0
176  signal bf_0_ctrl_cnt_out:std_logic_vector(3 downto 0);
177  signal bf_0_out_0_r,bf_0_out_0_i,bf_0_out_1_r,bf_0_out_1_i:
178     std_logic_vector(ifft_in_ws-1+1 downto 0);
179  ----------SIGNALS FOR STAGE 1-----------------------------------------
180  ----------signals from delay #0
181  signal del_0_ctrl_cnt_out:std_logic_vector(3 downto 0);
182  signal del_0_out_0_r,del_0_out_0_i,del_0_out_1_r,del_0_out_1_i:
183     std_logic_vector(ifft_in_ws-1+1 downto 0);
184  ----------signals from commutator #0
185  signal com_0_ctrl_cnt_out:std_logic_vector(3 downto 0);
186  signal com_0_out_0_r,com_0_out_0_i,com_0_out_1_r,com_0_out_1_i:
187     std_logic_vector(ifft_in_ws-1+1 downto 0);
188  ----------signals from delay #1
189  signal del_1_ctrl_cnt_out:std_logic_vector(3 downto 0);
190  signal del_1_out_0_r,del_1_out_0_i,del_1_out_1_r,del_1_out_1_i:
191     std_logic_vector(ifft_in_ws-1+1 downto 0);
192  ----------signals from multiplier #0
193  signal mul_0_ctrl_cnt_out:std_logic_vector(3 downto 0);
194  signal mul_0_out_0_r,mul_0_out_0_i,mul_0_out_1_r,mul_0_out_1_i:
195     std_logic_vector(ifft_in_ws-1+1 downto 0);
196  ----------signals from butterfly #1
197  signal bf_1_ctrl_cnt_out:std_logic_vector(3 downto 0);
198  signal bf_1_out_0_r,bf_1_out_0_i,bf_1_out_1_r,bf_1_out_1_i:
199     std_logic_vector(ifft_in_ws-1+2 downto 0);
200  ----------SIGNALS FOR STAGE 2-----------------------------------------
201  ----------signals from delay #2
202  signal del_2_ctrl_cnt_out:std_logic_vector(3 downto 0);
203  signal del_2_out_0_r,del_2_out_0_i,del_2_out_1_r,del_2_out_1_i:
204     std_logic_vector(ifft_in_ws-1+2 downto 0);
205  ----------signals from commutator #1
206  signal com_1_ctrl_cnt_out:std_logic_vector(3 downto 0);
207  signal com_1_out_0_r,com_1_out_0_i,com_1_out_1_r,com_1_out_1_i:
208     std_logic_vector(ifft_in_ws-1+2 downto 0);
209  ----------signals from delay #3
210  signal del_3_ctrl_cnt_out:std_logic_vector(3 downto 0);
211  signal del_3_out_0_r,del_3_out_0_i,del_3_out_1_r,del_3_out_1_i:
212     std_logic_vector(ifft_in_ws-1+2 downto 0);
213  ----------signals from multiplier #1
214  signal mul_1_ctrl_cnt_out:std_logic_vector(3 downto 0);
215  signal mul_1_out_0_r,mul_1_out_0_i,mul_1_out_1_r,mul_1_out_1_i:
216     std_logic_vector(ifft_in_ws-1+2 downto 0);
217  ----------signals from butterfly #2
218  signal bf_2_ctrl_cnt_out:std_logic_vector(3 downto 0);
219  signal bf_2_out_0_r,bf_2_out_0_i,bf_2_out_1_r,bf_2_out_1_i:
220     std_logic_vector(ifft_in_ws-1+3 downto 0);
221  ----------SIGNALS FOR STAGE 3-----------------------------------------
222  ----------signals from delay #4
223  signal del_4_ctrl_cnt_out:std_logic_vector(3 downto 0);
224  signal del_4_out_0_r,del_4_out_0_i,del_4_out_1_r,del_4_out_1_i:
225     std_logic_vector(ifft_in_ws-1+3 downto 0);
226  ----------signals from commutator #2
227  signal com_2_ctrl_cnt_out:std_logic_vector(3 downto 0);
228  signal com_2_out_0_r,com_2_out_0_i,com_2_out_1_r,com_2_out_1_i:
```

```
229      std_logic_vector(ifft_in_ws-1+3 downto 0);
230    ----------signals from delay #5
231    signal del_5_ctrl_cnt_out:std_logic_vector(3 downto 0);
232    signal del_5_out_0_r,del_5_out_0_i,del_5_out_1_r,del_5_out_1_i:
233      std_logic_vector(ifft_in_ws-1+3 downto 0);
234    ----------signals from multiplier #2
235    signal mul_2_ctrl_cnt_out:std_logic_vector(3 downto 0);
236    signal mul_2_out_0_r,mul_2_out_0_i,mul_2_out_1_r,mul_2_out_1_i:
237      std_logic_vector(ifft_in_ws-1+3 downto 0);
238    ----------signals from butterfly #3
239    signal bf_3_ctrl_cnt_out:std_logic_vector(3 downto 0);
240    signal bf_3_out_0_r,bf_3_out_0_i,bf_3_out_1_r,bf_3_out_1_i:
241      std_logic_vector(ifft_in_ws-1+4 downto 0);
242
243
244
245      begin
246      ifft_inp_reg_1:ifft_inp_reg
247        generic map(
248          wordsize => ifft_in_ws)
249        port map(
250          SR_clk => clk,
251          SR_rdy_in => rdy_in,
252          SR_input_real => ifft_data_in_real,
253          SR_input_imag => ifft_data_in_imag,
254          SR_data_out_0_real => inp_SR_out_0_r,
255          SR_data_out_0_imag => inp_SR_out_0_i,
256          SR_data_out_1_real => inp_SR_out_1_r,
257          SR_data_out_1_imag => inp_SR_out_1_i,
258          SR_ctrl_cnt_out => inp_SR_ctrl_cnt_out);
259
260    --stage 0 just one butterfly, always required except for trivial (1 pt FFT)
261     G0:if log(nmb_of_fft_pt) > 0 generate
262        butterfly_0:butterfly
263            generic map(
264                wordsize => ifft_in_ws)
265            port map(
266                Btf_clk => clk,
267                      Btf_ctrl_cnt_in => inp_SR_ctrl_cnt_out,
268                      Btf_in_0_real => inp_SR_out_0_r,
269                      Btf_in_0_imag => inp_SR_out_0_i,
270                      Btf_in_1_real => inp_SR_out_1_r,
271                      Btf_in_1_imag => inp_SR_out_1_i,
272                      Btf_out_0_real => bf_0_out_0_r,
273                      Btf_out_0_imag => bf_0_out_0_i,
274                      Btf_out_1_real => bf_0_out_1_r,
275                      Btf_out_1_imag => bf_0_out_1_i,
276                      Btf_ctrl_cnt_out => bf_0_ctrl_cnt_out);
277        end generate;
278    -----------------stage 1, neccessary if number of FFT pts > 2-------------
279     G1:if log(nmb_of_FFT_pt)-1 > 0 generate
280
281        delay_0:delay
282            generic map(
283                stage => 2**(abs(log(nmb_of_FFT_pt)-2)),--abs only to make
284                                                        --compiler quiet
285                delayed_path => 1,
```

```
286             wordsize => ifft_in_ws+1)
287         port map(
288             Del_clk => clk,
289             Del_ctrl_cnt_in => bf_0_ctrl_cnt_out,
290                   Del_data_in_0_real => bf_0_out_0_r,
291                   Del_data_in_0_imag => bf_0_out_0_i,
292                   Del_data_in_1_real => bf_0_out_1_r,
293                   Del_data_in_1_imag => bf_0_out_1_i,
294                   Del_data_out_0_real => del_0_out_0_r,
295                   Del_data_out_0_imag => del_0_out_0_i,
296                   Del_data_out_1_real => del_0_out_1_r,
297                   Del_data_out_1_imag => del_0_out_1_i,
298                   Del_ctrl_cnt_out => del_0_ctrl_cnt_out);
299
300     fft_comm_0:fft_comm
301         generic map(
302             wordsize => ifft_in_ws+1,
303             controlbit => log(nmb_of_FFT_pt)-2)  --0 to 3, determines which
304             --bit of the control counter switches the commutator
305         port map(
306             Com_clk => clk,
307             Com_ctrl_cnt_in => del_0_ctrl_cnt_out,
308                   Com_in_0_real => del_0_out_0_r,
309             Com_in_0_imag => del_0_out_0_i,
310                   Com_in_1_real => del_0_out_1_r,
311                   Com_in_1_imag => del_0_out_1_i,
312                   Com_out_0_real => com_0_out_0_r,
313                   Com_out_0_imag => com_0_out_0_i,
314                   Com_out_1_real => com_0_out_1_r,
315                   Com_out_1_imag => com_0_out_1_i,
316                   Com_ctrl_cnt_out => com_0_ctrl_cnt_out);
317
318     delay_1:delay
319         generic map(
320             stage => 2**(abs(log(nmb_of_FFT_pt)-2)),
321             delayed_path => 0,
322             wordsize => ifft_in_ws+1)
323         port map(
324             Del_clk => clk,
325             Del_ctrl_cnt_in => com_0_ctrl_cnt_out,
326                   Del_data_in_0_real => com_0_out_0_r,
327                   Del_data_in_0_imag => com_0_out_0_i,
328                   Del_data_in_1_real => com_0_out_1_r,
329                   Del_data_in_1_imag => com_0_out_1_i,
330                   Del_data_out_0_real => del_1_out_0_r,
331                   Del_data_out_0_imag => del_1_out_0_i,
332                   Del_data_out_1_real => del_1_out_1_r,
333                   Del_data_out_1_imag => del_1_out_1_i,
334                   Del_ctrl_cnt_out => del_1_ctrl_cnt_out);
335
336     mul2val_0:mul2val
337         generic map(
338             wordsize => ifft_in_ws+1,
339             controlbit => log(nmb_of_FFT_pt)-2)  --0 to 3, determines which
340             --bit of the control counter selects the factor (1 or j)
341         port map(
342             M2_clk => clk,
```

```
343                     M2_ctrl_cnt_in => del_1_ctrl_cnt_out,
344                     M2_in_0_real => del_1_out_0_r,
345                     M2_in_0_imag => del_1_out_0_i,
346                     M2_in_1_real => del_1_out_1_r,
347                     M2_in_1_imag => del_1_out_1_i,
348                     M2_out_0_real => mul_0_out_0_r,
349                     M2_out_0_imag => mul_0_out_0_i,
350                     M2_out_1_real => mul_0_out_1_r,
351                     M2_out_1_imag => mul_0_out_1_i,
352                     M2_ctrl_cnt_out => mul_0_ctrl_cnt_out);
353
354     butterfly_1:butterfly
355         generic map(
356             wordsize => ifft_in_ws+1)
357         port map(
358             Btf_clk => clk,
359                     Btf_ctrl_cnt_in => mul_0_ctrl_cnt_out,
360                     Btf_in_0_real => mul_0_out_0_r,
361                     Btf_in_0_imag => mul_0_out_0_i,
362                     Btf_in_1_real => mul_0_out_1_r,
363                     Btf_in_1_imag => mul_0_out_1_i,
364                     Btf_out_0_real => bf_1_out_0_r,
365                     Btf_out_0_imag => bf_1_out_0_i,
366                     Btf_out_1_real => bf_1_out_1_r,
367                     Btf_out_1_imag => bf_1_out_1_i,
368                     Btf_ctrl_cnt_out => bf_1_ctrl_cnt_out);
369
370     end generate;
371 ----------------stage 2, necessary if number of FFT pts > 4----------------
372  G2:if log(nmb_of_FFT_pt)-2 > 0 generate
373
374     delay_2:delay
375         generic map(
376             stage => 2**(abs(log(nmb_of_FFT_pt)-3)),
377             delayed_path => 1,
378             wordsize => ifft_in_ws+2)
379         port map(
380             Del_clk => clk,
381             Del_ctrl_cnt_in => bf_1_ctrl_cnt_out,
382                     Del_data_in_0_real => bf_1_out_0_r,
383                     Del_data_in_0_imag => bf_1_out_0_i,
384                     Del_data_in_1_real => bf_1_out_1_r,
385                     Del_data_in_1_imag => bf_1_out_1_i,
386                     Del_data_out_0_real => del_2_out_0_r,
387                     Del_data_out_0_imag => del_2_out_0_i,
388                     Del_data_out_1_real => del_2_out_1_r,
389                     Del_data_out_1_imag => del_2_out_1_i,
390                     Del_ctrl_cnt_out => del_2_ctrl_cnt_out);
391
392     fft_comm_1:fft_comm
393         generic map(
394             wordsize => ifft_in_ws+2,
395             controlbit => log(nmb_of_FFT_pt)-3)  --0 to 3, determines which
396             --bit of the control counter switches the commutator
397         port map(
398             Com_clk => clk,
399             Com_ctrl_cnt_in => del_2_ctrl_cnt_out,
```

```
400                 Com_in_0_real => del_2_out_0_r,
401             Com_in_0_imag => del_2_out_0_i,
402                 Com_in_1_real => del_2_out_1_r,
403                 Com_in_1_imag => del_2_out_1_i,
404                 Com_out_0_real => com_1_out_0_r,
405                 Com_out_0_imag => com_1_out_0_i,
406                 Com_out_1_real => com_1_out_1_r,
407                 Com_out_1_imag => com_1_out_1_i,
408                 Com_ctrl_cnt_out => com_1_ctrl_cnt_out);
409
410     delay_3:delay
411         generic map(
412             stage => 2**(abs(log(nmb_of_FFT_pt)-3)),
413             delayed_path => 0,
414             wordsize => ifft_in_ws+2)
415         port map(
416             Del_clk => clk,
417             Del_ctrl_cnt_in => com_1_ctrl_cnt_out,
418                 Del_data_in_0_real => com_1_out_0_r,
419                 Del_data_in_0_imag => com_1_out_0_i,
420                 Del_data_in_1_real => com_1_out_1_r,
421                 Del_data_in_1_imag => com_1_out_1_i,
422                 Del_data_out_0_real => del_3_out_0_r,
423                 Del_data_out_0_imag => del_3_out_0_i,
424                 Del_data_out_1_real => del_3_out_1_r,
425                 Del_data_out_1_imag => del_3_out_1_i,
426                 Del_ctrl_cnt_out => del_3_ctrl_cnt_out);
427
428     mul4val_0:mul4val
429         generic map(
430             ws => ifft_in_ws+2,
431             ctrl_word_LSB => log(nmb_of_FFT_pt)-3)
432         port map(
433             M4_clk => clk,
434                 M4_ctrl_cnt_in => del_3_ctrl_cnt_out,
435                 M4_in_0_real => del_3_out_0_r,
436                 M4_in_0_imag => del_3_out_0_i,
437                 M4_in_1_real => del_3_out_1_r,
438                 M4_in_1_imag => del_3_out_1_i,
439                 M4_out_0_real => mul_1_out_0_r,
440                 M4_out_0_imag => mul_1_out_0_i,
441                 M4_out_1_real => mul_1_out_1_r,
442                 M4_out_1_imag => mul_1_out_1_i,
443                 M4_ctrl_cnt_out => mul_1_ctrl_cnt_out);
444
445     butterfly_2:butterfly
446         generic map(
447             wordsize => ifft_in_ws+2)
448         port map(
449             Btf_clk => clk,
450                 Btf_ctrl_cnt_in => mul_1_ctrl_cnt_out,
451                 Btf_in_0_real => mul_1_out_0_r,
452                 Btf_in_0_imag => mul_1_out_0_i,
453                 Btf_in_1_real => mul_1_out_1_r,
454                 Btf_in_1_imag => mul_1_out_1_i,
455                 Btf_out_0_real => bf_2_out_0_r,
456                 Btf_out_0_imag => bf_2_out_0_i,
```

```
457                     Btf_out_1_real => bf_2_out_1_r,
458                     Btf_out_1_imag => bf_2_out_1_i,
459                     Btf_ctrl_cnt_out => bf_2_ctrl_cnt_out);
460
461     end generate;
462 ---------------stage 3, necessary if number of FFT pts > 8----------------
463  G3:if log(nmb_of_FFT_pt)-3 > 0 generate
464
465     delay_4:delay
466         generic map(
467             stage => 2**(abs(log(nmb_of_FFT_pt)-4)),
468             delayed_path => 1,
469             wordsize => ifft_in_ws+3)
470         port map(
471             Del_clk => clk,
472             Del_ctrl_cnt_in => bf_2_ctrl_cnt_out,
473                   Del_data_in_0_real => bf_2_out_0_r,
474                   Del_data_in_0_imag => bf_2_out_0_i,
475                   Del_data_in_1_real => bf_2_out_1_r,
476                   Del_data_in_1_imag => bf_2_out_1_i,
477                   Del_data_out_0_real => del_4_out_0_r,
478                   Del_data_out_0_imag => del_4_out_0_i,
479                   Del_data_out_1_real => del_4_out_1_r,
480                   Del_data_out_1_imag => del_4_out_1_i,
481                   Del_ctrl_cnt_out => del_4_ctrl_cnt_out);
482
483     fft_comm_2:fft_comm
484         generic map(
485             wordsize => ifft_in_ws+3,
486             controlbit => log(nmb_of_FFT_pt)-4)  --0 to 3, determines which
487             --bit of the control counter switches the commutator
488         port map(
489             Com_clk => clk,
490             Com_ctrl_cnt_in => del_4_ctrl_cnt_out,
491                   Com_in_0_real => del_4_out_0_r,
492             Com_in_0_imag => del_4_out_0_i,
493                   Com_in_1_real => del_4_out_1_r,
494                   Com_in_1_imag => del_4_out_1_i,
495                   Com_out_0_real => com_2_out_0_r,
496                   Com_out_0_imag => com_2_out_0_i,
497                   Com_out_1_real => com_2_out_1_r,
498                   Com_out_1_imag => com_2_out_1_i,
499                   Com_ctrl_cnt_out => com_2_ctrl_cnt_out);
500
501     delay_5:delay
502         generic map(
503             stage => 2**(abs(log(nmb_of_FFT_pt)-4)),
504             delayed_path => 0,
505             wordsize => ifft_in_ws+3)
506         port map(
507             Del_clk => clk,
508             Del_ctrl_cnt_in => com_2_ctrl_cnt_out,
509                   Del_data_in_0_real => com_2_out_0_r,
510                   Del_data_in_0_imag => com_2_out_0_i,
511                   Del_data_in_1_real => com_2_out_1_r,
512                   Del_data_in_1_imag => com_2_out_1_i,
513                   Del_data_out_0_real => del_5_out_0_r,
```

```
514                     Del_data_out_0_imag => del_5_out_0_i,
515                     Del_data_out_1_real => del_5_out_1_r,
516                     Del_data_out_1_imag => del_5_out_1_i,
517                     Del_ctrl_cnt_out => del_5_ctrl_cnt_out);
518
519       mul8val_0:mul8val
520           generic map(
521               ws => ifft_in_ws+3,
522               ctrl_word_LSB => log(nmb_of_FFT_pt)-4)
523           port map(
524               M8_clk => clk,
525                     M8_ctrl_cnt_in => del_5_ctrl_cnt_out,
526                     M8_in_0_real => del_5_out_0_r,
527                     M8_in_0_imag => del_5_out_0_i,
528                     M8_in_1_real => del_5_out_1_r,
529                     M8_in_1_imag => del_5_out_1_i,
530                     M8_out_0_real => mul_2_out_0_r,
531                     M8_out_0_imag => mul_2_out_0_i,
532                     M8_out_1_real => mul_2_out_1_r,
533                     M8_out_1_imag => mul_2_out_1_i,
534                     M8_ctrl_cnt_out => mul_2_ctrl_cnt_out);
535
536       butterfly_3:butterfly
537           generic map(
538               wordsize => ifft_in_ws+3)
539           port map(
540               Btf_clk => clk,
541                     Btf_ctrl_cnt_in => mul_2_ctrl_cnt_out,
542                     Btf_in_0_real => mul_2_out_0_r,
543                     Btf_in_0_imag => mul_2_out_0_i,
544                     Btf_in_1_real => mul_2_out_1_r,
545                     Btf_in_1_imag => mul_2_out_1_i,
546                     Btf_out_0_real => bf_3_out_0_r,
547                     Btf_out_0_imag => bf_3_out_0_i,
548                     Btf_out_1_real => bf_3_out_1_r,
549                     Btf_out_1_imag => bf_3_out_1_i,
550                     Btf_ctrl_cnt_out => bf_3_ctrl_cnt_out);
551
552       end generate;
553
554   --Now decide which signal connect to the output shitf register-----------
555    G4:if log(nmb_of_FFT_pt)-1 = 0 generate --2 pt FFT, just 1 butterfly
556       ifft_out_reg_1:ifft_out_reg
557           generic map(
558               wordsize => ifft_in_ws+1)
559           port map(
560               SR_clk => clk,
561               SR_ctrl_cnt_in => bf_0_ctrl_cnt_out,
562               SR_data_in_0_real => bf_0_out_0_r,
563                     SR_data_in_0_imag => bf_0_out_0_i,
564                     SR_data_in_1_real => bf_0_out_1_r,
565                     SR_data_in_1_imag => bf_0_out_1_i,
566                     SR_rdy_out => ifft_rdy_out,
567                     SR_output_real => ifft_data_out_real,
568                     SR_output_imag => ifft_data_out_imag);
569       end generate;
570
```

```
571    G5:if log(nmb_of_FFT_pt)-2 = 0 generate
572       ifft_out_reg_2:ifft_out_reg
573           generic map(
574               wordsize => ifft_in_ws+2)
575           port map(
576               SR_clk => clk,
577               SR_ctrl_cnt_in => bf_1_ctrl_cnt_out,
578                     SR_data_in_0_real => bf_1_out_0_r,
579                     SR_data_in_0_imag => bf_1_out_0_i,
580                     SR_data_in_1_real => bf_1_out_1_r,
581                     SR_data_in_1_imag => bf_1_out_1_i,
582                     SR_rdy_out => ifft_rdy_out,
583                     SR_output_real => ifft_data_out_real,
584                     SR_output_imag => ifft_data_out_imag);
585       end generate;
586
587    G6:if log(nmb_of_FFT_pt)-3 = 0 generate
588       ifft_out_reg_3:ifft_out_reg
589           generic map(
590               wordsize => ifft_in_ws+3)
591           port map(
592               SR_clk => clk,
593               SR_ctrl_cnt_in => bf_2_ctrl_cnt_out,
594                     SR_data_in_0_real => bf_2_out_0_r,
595                     SR_data_in_0_imag => bf_2_out_0_i,
596                     SR_data_in_1_real => bf_2_out_1_r,
597                     SR_data_in_1_imag => bf_2_out_1_i,
598                     SR_rdy_out => ifft_rdy_out,
599                     SR_output_real => ifft_data_out_real,
600                     SR_output_imag => ifft_data_out_imag);
601       end generate;
602
603    G7:if log(nmb_of_FFT_pt)-4 = 0 generate
604       ifft_out_reg_4:ifft_out_reg
605           generic map(
606               wordsize => ifft_in_ws+4)
607           port map(
608               SR_clk => clk,
609               SR_ctrl_cnt_in => bf_3_ctrl_cnt_out,
610                     SR_data_in_0_real => bf_3_out_0_r,
611                     SR_data_in_0_imag => bf_3_out_0_i,
612                     SR_data_in_1_real => bf_3_out_1_r,
613                     SR_data_in_1_imag => bf_3_out_1_i,
614                     SR_rdy_out => ifft_rdy_out,
615                     SR_output_real => ifft_data_out_real,
616                     SR_output_imag => ifft_data_out_imag);
617       end generate;
618
619    end ifft_arch;
620
```

```
1  ------------------------------------------------------------------------
2  ------------------------------inp_sr_i.vhd------------------------------
3  ------------------------------------------------------------------------
4  --Describes the input shift register of the pipeline IFFT, same as for the
5  --FFT except for the smaller word size. That's the reason for repeating
6  --this file. Remember, the wordsizes are determined by the definition of
7  --data types in fltb_par.vhd
8
9  library IEEE;
10 use IEEE.STD_LOGIC_1164.all;
11 use IEEE.STD_LOGIC_arith.all;
12 use IEEE.STD_LOGIC_signed.all;
13
14 library work;
15 use work.fltb_par.all;
16
17 entity ifft_inp_reg is
18   generic(
19       wordsize:integer);
20   port(
21       SR_clk:in std_logic;
22       SR_rdy_in:in std_logic;
23       SR_input_real:in ifft_in_data;          --array
24       SR_input_imag:in ifft_in_data;          --array
25       SR_data_out_0_real:out std_logic_vector(wordsize-1 downto 0);
26       SR_data_out_0_imag:out std_logic_vector(wordsize-1 downto 0);
27       SR_data_out_1_real:out std_logic_vector(wordsize-1 downto 0);
28       SR_data_out_1_imag:out std_logic_vector(wordsize-1 downto 0);
29       SR_ctrl_cnt_out:out std_logic_vector(3 downto 0));
30 end ifft_inp_reg;
31
32
33
34 architecture ifft_inp_reg_arch of ifft_inp_reg is
35
36 signal reg_real,reg_imag:ifft_in_data;     --array
37 signal ctrl_cnt:std_logic_vector(3 downto 0);
38
39 begin
40
41
42 process(SR_clk)
43     begin
44     if SR_CLK'event and SR_CLK ='1' then
45         if SR_rdy_in = '1' then
46             reg_real <= SR_input_real;
47             reg_imag <= SR_input_imag;
48             ctrl_cnt <= "0000";
49         else
50             ctrl_cnt <= ctrl_cnt + CONV_STD_LOGIC_VECTOR(1,4);
51             if nmb_of_FFT_pt > 2 then
52                L1:for i in 0 to abs(nmb_of_FFT_pt/2-2) loop --abs only
53                   --used to get it compiled
54                   reg_real(i) <= reg_real(i+1);
55                   reg_imag(i) <= reg_imag(i+1);
56                end loop L1;
57             end if;
```

```
58              reg_real(nmb_of_FFT_pt/2-1) <=
59  CONV_STD_LOGIC_VECTOR(0,ifft_in_ws);
60              reg_imag(nmb_of_FFT_pt/2-1) <=
61  CONV_STD_LOGIC_VECTOR(0,ifft_in_ws);
62              L2:for i in nmb_of_FFT_pt/2 to nmb_of_FFT_pt-2 loop
63                 reg_real(i) <= reg_real(i+1);
64                 reg_imag(i) <= reg_imag(i+1);
65              end loop L2;
66              reg_real(nmb_of_FFT_pt-1) <=
67  CONV_STD_LOGIC_VECTOR(0,ifft_in_ws);
68              reg_imag(nmb_of_FFT_pt-1) <=
69  CONV_STD_LOGIC_VECTOR(0,ifft_in_ws);
70          end if;
71      end if;
72
73  end process;
74
75
76  process(reg_real,reg_imag,ctrl_cnt)
77      begin
78      SR_data_out_0_real <= reg_real(0);
79      SR_data_out_0_imag <= reg_imag(0);
80      SR_data_out_1_real <= reg_real(nmb_of_FFT_pt/2);
81      SR_data_out_1_imag <= reg_imag(nmb_of_FFT_pt/2);
82      SR_ctrl_cnt_out <= ctrl_cnt;
83  end process;
84
85
86  end ifft_inp_reg_arch;
87
```

```
1  --------------------------------------------------------------------------
2  ------------------------------outp_sr_i.vhd-------------------------------
3  --------------------------------------------------------------------------
4  --Describes the output shift register of the pipeline IFFT, same as for the
5  --FFT except for the smaller word size. That's the reason for repeating
6  --this file. Remember, the wordsizes are determined by the definition of
7  --data types in fltb_par.vhd
8
9
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.all;
12 use IEEE.STD_LOGIC_arith.all;
13 use IEEE.STD_LOGIC_signed.all;
14
15 library work;
16 use work.fltb_par.all;
17
18 entity ifft_out_reg is
19   generic(
20       wordsize:integer);
21   port(
22       SR_clk:in std_logic;
23       SR_ctrl_cnt_in:in std_logic_vector(3 downto 0);
24       SR_data_in_0_real:in std_logic_vector(wordsize-1 downto 0);
25       SR_data_in_0_imag:in std_logic_vector(wordsize-1 downto 0);
26       SR_data_in_1_real:in std_logic_vector(wordsize-1 downto 0);
27       SR_data_in_1_imag:in std_logic_vector(wordsize-1 downto 0);
28       SR_rdy_out:out std_logic;
29       SR_output_real:out ifft_out_data;          --array
30       SR_output_imag:out ifft_out_data);          --array
31 end ifft_out_reg;
32
33
34
35 architecture ifft_out_reg_arch of ifft_out_reg is
36
37 signal reg_real,reg_imag:ifft_out_data;      --array
38
39 begin
40
41 process(SR_clk)
42
43     begin
44
45     if SR_CLK'event and SR_CLK ='1' then
46         if SR_ctrl_cnt_in = CONV_STD_LOGIC_VECTOR(nmb_of_FFT_pt/2-1,4) then
47             SR_rdy_out <= '1';
48         else
49             SR_rdy_out <= '0';
50         end if;
51      if nmb_of_FFT_pt > 2 then
52         L1:for i in 0 to abs(nmb_of_FFT_pt/2-2) loop --abs to get
53                                         --it compiled
54             reg_real(i) <= reg_real(i+1);
55             reg_imag(i) <= reg_imag(i+1);
56            end loop L1;
57      end if;
```

```
58          reg_real(nmb_of_FFT_pt/2-1) <= SR_data_in_0_real;
59          reg_imag(nmb_of_FFT_pt/2-1) <= SR_data_in_0_imag;
60       L2:for i in nmb_of_FFT_pt/2 to nmb_of_FFT_pt-2 loop
61              reg_real(i) <= reg_real(i+1);
62              reg_imag(i) <= reg_imag(i+1);
63          end loop L2;
64          reg_real(nmb_of_FFT_pt-1) <= SR_data_in_1_real;
65          reg_imag(nmb_of_FFT_pt-1) <= SR_data_in_1_imag;
66
67      end if;
68
69  end process;
70
71  process(reg_real,reg_imag)   --reorder output sequence
72      constant rev_index:int_array:=
73  --                (0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,        --for 2 pt FFT
74  --                 0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3,        --for 4 pt FFT
75  --                 0,2,1,3,4,6,5,7,0,2,1,3,4,6,5,7,        --for 8 pt FFT
76  --                 0,4,2,6,1,5,3,7,8,12,10,14,9,13,11,15); --for 16 pt FFT
77
78                    (0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,        --for 2 pt FFT
79                     0,3,2,1,0,3,2,1,0,3,2,1,0,3,2,1,        --for 4 pt FFT
80                     0,7,5,6,4,3,1,2,0,7,5,6,4,3,1,2,        --for 8 pt FFT
81                     0,15,11,13,9,14,10,12,8,7,3,5,1,6,2,4); --for 16 pt FFT
82
83      --Why is MSB not involved in the bit reversal scheme ? Look book
84      --Oppenheim, p.597, fig. 9.14: The output of the upper butterfly branch
85      --has always index with MSB cleared, the output of the lower branch
86      --has always the MSB set.
87
88      begin
89  L10:for i in 0 to nmb_of_FFT_pt-1 loop
90          SR_output_real(i) <= reg_real(rev_index((log(nmb_of_FFT_pt)-
91  1)*16+i));
92          SR_output_imag(i) <= reg_imag(rev_index((log(nmb_of_FFT_pt)-
93  1)*16+i));
94
95      end loop L10;
96
97  end process;
98
99  end ifft_out_reg_arch;
100
101
```

```
1  ----------------------------------------------------------------------------
2  --------------------------------sy_fil4.vhd-------------------------------
3  ----------------------------------------------------------------------------
4  --This file puts 4 filters, generated by CORE GENERATOR together
5  --Similar files are created for the 8 and 16 subband analyzer
6
7
8  library IEEE;
9  use IEEE.STD_LOGIC_1164.all;
10 use IEEE.STD_LOGIC_arith.all;
11 use IEEE.STD_LOGIC_signed.all;
12
13 library Work;
14 use Work.fltb_par.all;
15
16 entity Sy_fil is
17   port(
18      Clk:in STD_LOGIC;
19      Reset:in STD_LOGIC;
20      Sy_fil_data_in:in Sy_fil_in_data;
21      Sy_fil_data_out:out Sy_fil_out_data;
22      Sy_fil_rdy_out:out STD_LOGIC);
23 end Sy_fil;
24
25 architecture Sy_fil_arch of Sy_fil is
26
27 component sy4_pp1 port (
28       data: IN std_logic_VECTOR(7 downto 0);
29       nd: IN std_logic;
30       rfd: OUT std_logic;
31       sinf: IN std_logic;
32       sinr: IN std_logic;
33       soutf: OUT std_logic;
34       soutr: OUT std_logic;
35       ck: IN std_logic;
36       rslt: OUT std_logic_VECTOR(Sy_filt_out_ws-1 downto 0);
37       rdy: OUT std_logic);
38 end component;
39
40 component sy4_pp2 port (
41       data: IN std_logic_VECTOR(7 downto 0);
42       nd: IN std_logic;
43       rfd: OUT std_logic;
44       sinf: IN std_logic;
45       sinr: IN std_logic;
46       soutf: OUT std_logic;
47       soutr: OUT std_logic;
48       ck: IN std_logic;
49       rslt: OUT std_logic_VECTOR(Sy_filt_out_ws-1 downto 0);
50       rdy: OUT std_logic);
51 end component;
52
53 component sy4_pp3 port (
54       data: IN std_logic_VECTOR(7 downto 0);
55       nd: IN std_logic;
56       rfd: OUT std_logic;
57       sinf: IN std_logic;
```

```
58          sinr: IN std_logic;
59          soutf: OUT std_logic;
60          soutr: OUT std_logic;
61          ck: IN std_logic;
62          rslt: OUT std_logic_VECTOR(Sy_filt_out_ws-1 downto 0);
63          rdy: OUT std_logic);
64    end component;
65
66    component sy4_pp4 port (
67          data: IN std_logic_VECTOR(7 downto 0);
68          nd: IN std_logic;
69          rfd: OUT std_logic;
70          sinf: IN std_logic;
71          sinr: IN std_logic;
72          soutf: OUT std_logic;
73          soutr: OUT std_logic;
74          ck: IN std_logic;
75          rslt: OUT std_logic_VECTOR(Sy_filt_out_ws-1 downto 0);
76          rdy: OUT std_logic);
77    end component;
78
79
80
81       signal nd,rfd1,rfd2,rfd3,rfd4:std_logic;
82       signal sinf1,sinf2,sinf3,sinf4,sinr1,sinr2,sinr3,sinr4:std_logic;
83       signal soutf1,soutf2,soutf3,soutf4,soutr1,soutr2,soutr3,soutr4:std_logic;
84       signal d1,d2,d3:std_logic;
85
86       begin
87
88    sy4_pp1_1 : sy4_pp1 port map (
89          data => Sy_fil_data_in(0),
90          nd => nd,
91          rfd => rfd1,
92          sinf => sinf1,
93          sinr => sinr1,
94          soutf => soutf1,
95          soutr => soutr1,
96          ck => Clk,
97          rslt => Sy_fil_data_out(0),
98          rdy => Sy_fil_rdy_out);
99
100   sy4_pp2_1 : sy4_pp2 port map (
101         data => Sy_fil_data_in(1),
102         nd => nd,
103         rfd => rfd2,
104         sinf => sinf2,
105         sinr => sinr2,
106         soutf => soutf2,
107         soutr => soutr2,
108         ck => Clk,
109         rslt => Sy_fil_data_out(1),
110         rdy => d1);
111
112   sy4_pp3_1 : sy4_pp3 port map (
113         data => Sy_fil_data_in(2),
114         nd => nd,
```

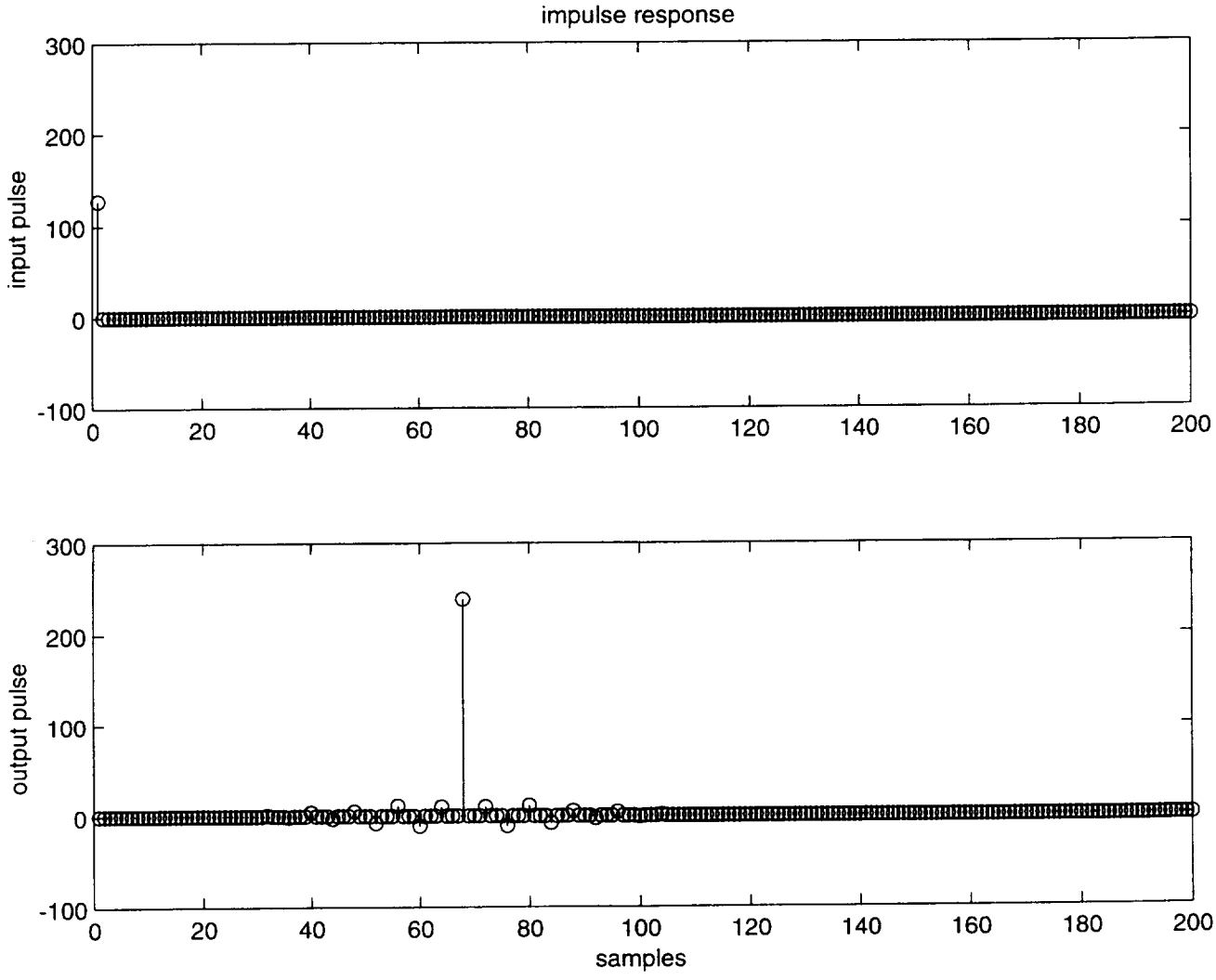
```
115         rfd => rfd3,
116         sinf => sinf3,
117         sinr => sinr3,
118         soutf => soutf3,
119         soutr => soutr3,
120         ck => Clk,
121         rslt => Sy_fil_data_out(2),
122         rdy => d2);
123
124   sy4_pp4_1 : sy4_pp4 port map (
125         data => Sy_fil_data_in(3),
126         nd => nd,
127         rfd => rfd4,
128         sinf => sinf4,
129         sinr => sinr4,
130         soutf => soutf4,
131         soutr => soutr4,
132         ck => Clk,
133         rslt => Sy_fil_data_out(3),
134         rdy =>d3);
135
136
137
138      process(rfd1,rfd2,rfd3,rfd4)
139      begin
140         nd <= rfd1 and rfd2 and rfd3 and rfd4;
141      end process;
142
143
144   end Sy_fil_arch;
145
146
```

```
1   ------------------------------dwsby2_n.vhd--------------------------------
2   --Describes the downsampler, used in the synthesis filter bank
3
4   library IEEE;
5   use IEEE.STD_LOGIC_1164.all;
6   use IEEE.STD_LOGIC_arith.all;
7   use IEEE.STD_LOGIC_signed.all;
8
9   library Work;
10  use Work.fltb_par.all;
11
12  entity dwsby2_n is
13    port(
14       Clk:in STD_LOGIC;
15       Reset:in STD_LOGIC;
16       dwsby2_data_in:in dwsby2_data;
17       dwsby2_data_out:out dwsby2_data;
18       downsampl_del:in STD_LOGIC);
19  end dwsby2_n;
20
21  architecture dwsby2_n_arch of dwsby2_n is
22
23     signal cnt:std_logic_vector(3 downto 0);
24     signal reg:dwsby2_data;
25
26     begin
27
28     process(Clk,Reset)
29     begin
30        if Reset = '0' then
31           cnt <= "0000";
32        elsif CLK'event and CLK ='1' then
33              cnt <= cnt + 1;
34              if downsampl_del = '1' then
35                 if cnt = "1000" then
36                    reg <= dwsby2_data_in;
37                 end if;
38              else
39                 if cnt = "0000" then
40                    reg <= dwsby2_data_in;
41                 end if;
42              end if;
43        end if;
44     end process;
45
46
47     process(reg)
48     begin
49        dwsby2_data_out <= reg;
50     end process;
51
52  end dwsby2_n_arch;
53
54
```

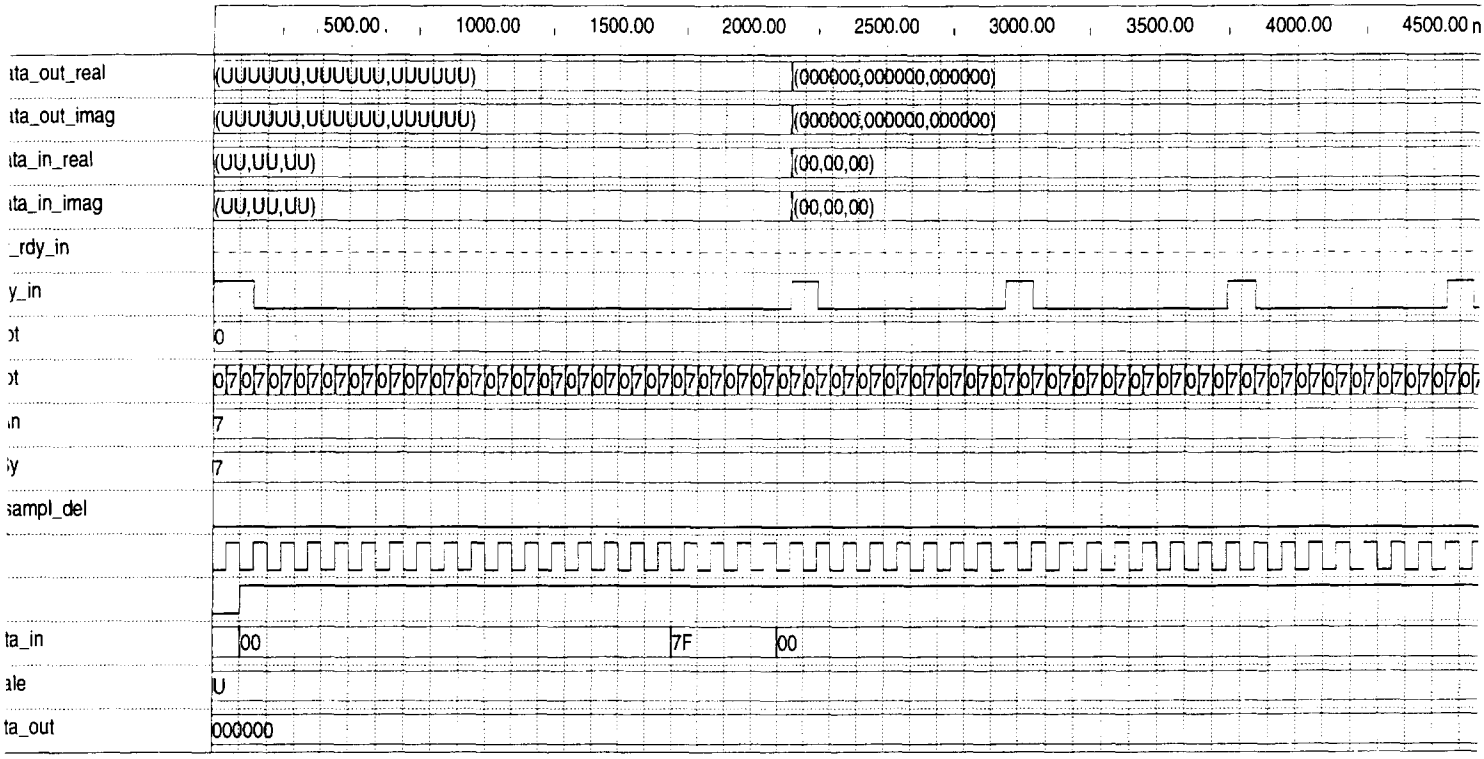
```
1  ------------------------------------------------------------------------
2  --------------------------com_sy_n.vhd-----------------------------------
3  ------------------------------------------------------------------------
4  --Describes the commutator, used in the synthesis filter bank
5
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  use IEEE.STD_LOGIC_arith.all;
9  use IEEE.STD_LOGIC_signed.all;
10
11 library Work;
12 use Work.fltb_par.all;
13
14 entity Com_Sy_n is
15    port(
16       Clk:in STD_LOGIC;
17       Reset:in STD_LOGIC;
18       Com_Sy_data_in:in dwsby2_data;
19       Com_Sy_data_out:out STD_LOGIC_VECTOR (Com_Sy_ws-1 downto 0));
20 end Com_Sy_n;
21
22 architecture Com_Sy_n_arch of Com_sy_n is
23
24    signal one_hot_cnt:std_logic_vector(15 downto 0);
25    signal outp_reg:STD_LOGIC_VECTOR (Com_Sy_ws-1 downto 0);
26
27    begin
28
29    process(Clk,Reset)
30    begin
31       if Reset = '0' then
32          one_hot_cnt <= "0000000000000001";
33       elsif (Clk'event and Clk = '1') then
34          one_hot_cnt(15 downto 1) <= one_hot_cnt(14 downto 0);
35          one_hot_cnt(0) <= one_hot_cnt(15);
36       end if;
37    end process;
38
39    process(Clk)
40    begin
41       if (Clk'event and Clk = '1') then
42       L1:for i in 0 to Nmb_of_subband-1 loop
43             if one_hot_cnt(16/Nmb_of_subband*i) = '1' then
44                outp_reg <= Com_Sy_data_in(CONV_INTEGER(CONV_STD_LOGIC_VECTOR
45                   (i+COMM_OFFSET,8) AND CONV_STD_LOGIC_VECTOR
46                      (Nmb_of_subband-1,8)));  --makes commutator offset
47                      --and wrap around, e.g. 2,3,0,1,2,3,...
48             end if;
49          end loop L1;
50       end if;
51    end process;
52
53    process(outp_reg)
54    begin
55       Com_Sy_data_out <= outp_reg;
56    end process;
57
```

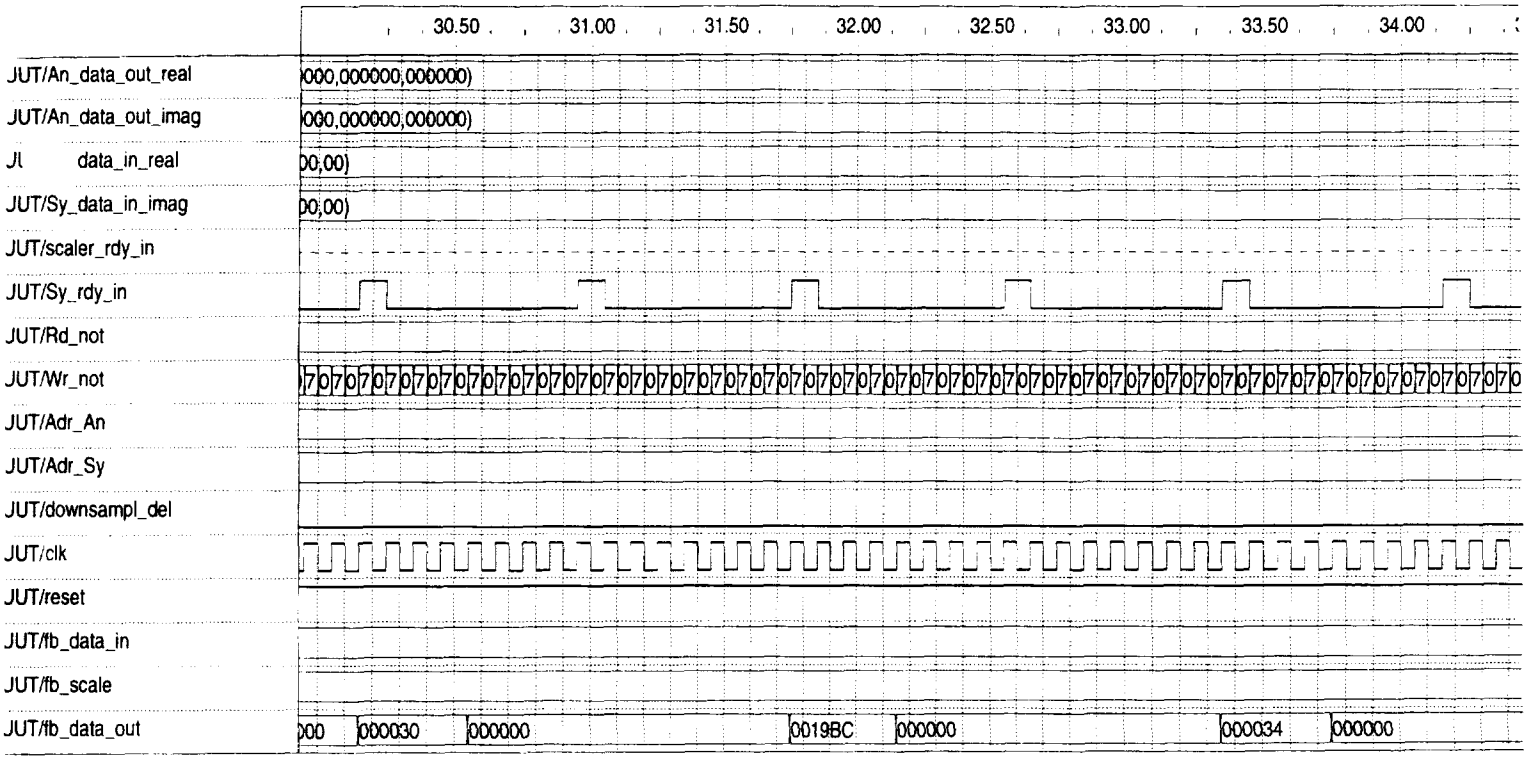
```
58    end Com_Sy_n_arch;
59
60
```

Appendix C
Impulse response, simulated with MATLAB


impulse response
input pulse
output pulse
samples
300
200
100
0
-100
0
20
40
60
80
100
120
140
160
180
200

Appendix D
Impulse response, simulated with ALDEC


500.00
1000.00
1500.00
2000.00
2500.00
3000.00
3500.00
4000.00
4500.00 n
ıta_out_real
(UUUUUU,UUUUUU,UUUUUU)
(000000,000000,000000)
ıta_out_imag
(UUUUUU,UUUUUU,UUUUUU)
(000000,000000,000000)
ıta_in_real
(UU,UU,UU)
(00,00,00)
ıta_in_imag
(UU,UU,UU)
(00,00,00)
_rdy_in
y_in
ɔt
0
ɔt
ın
7
ıy
7
ısampl_del
ta_in
00
7F
00
ıle
U
ta_out
000000


30.50
31.00
31.50
32.00
32.50
33.00
33.50
34.00
JUT/An_data_out_real
JUT/An_data_out_imag
Jl data_in_real
JUT/Sy_data_in_imag
JUT/scaler_rdy_in
JUT/Sy_rdy_in
JUT/Rd_not
JUT/Wr_not
JUT/Adr_An
JUT/Adr_Sy
JUT/downsampl_del
JUT/clk
JUT/reset
JUT/fb_data_in
JUT/fb_scale
JUT/fb_data_out
000030
000000
0019BC
000000
000034
000000

Appendix E
Motorola 56k code for pass through of subband signals

```
;*****************************sub_pass.asm*********************************
; Name: Stephan Berner                                          Date: 04/26/99
; SSN: [redacted]                                               Course: EE592
; Project: Final                                        Version: 2.0 (MPP 1.1)
; Modified to interface EVM to FPGAs for subband processing
;*****************************************************************************
; Purpose: Reading from and writing to external hardware
;
;-----------------------------------------------------------------------------
; Outline: This program reads a 24 bit input port and writes the data to
;          another 24 bit output port (pass through).
;          Operation:
;          1) External hardware writes a 24 bit word into a register and
;             generates a hardware interrupt (IRQ A).
;          2) DSP reads the data and writes it to an output register.
;             A "fast interrupt" is used, that means the command are written
;             directly into the vector table.
;             Alternatively, the input data could be written to an input
;             buffer, and an output buffer could could be written to the
;             output port
;          This program is based on pass.asm, all parts related to the
;          CODEC are removed.
;          Outline of the initialization:
;          1) Initialize PLL                (nothing changed)
;          2) Initialize external memory    (nothing changed)
;          3) Initialize stack              (nothing changed)
;          4) Disable CODEC interrupts      (new)
;          5) Initialize IRQ A              (new)
;          6) Set # of wait states          (new)
;
;
;          The routine "flash.asm" is included, which writes this code
;          and a boot routine into the EEPROM.
;          To store this program to the EEPROM, perform the following
;          steps:
;          1) Stop hardware interrupts by reseting the FPGA continuosly
;             (jumper between the reset pins)
;          2) Load this program
;          3) Set the program counter (PC) to 800h
;          4) Select "Run"
;          5) EVM should end at the instructions
;                  debug
;                  bra *
;          6) Reset EVM
;             Reset always in the following way:
;             6.1) Hold the FPGAs in the reset state by shortening the
;                  reset pin to GND. This keeps the IRQA pin, which is
;                  also a mode select pin, on a high level.
;             6.2) Now reset the EVM by cycling the power.
;             6.3) Start the FPGAs by releasing the reset pin.
;
;
;          For passing through real data (subband 0 and 2), the MSB from
;          the FPGA output will be read into bit 23 of x0, the LSB of the
;          FPGA output will be read into bit 8 of x0, bit 0 to bit 7 of
;          x0 have to be ignored.
;          For passing through complex data (subband 1), the MSB of the
```

```
58  ;          real part will be read to bit 23 of x0, the LSB will be read
59  ;          to bit 12 of x0, the MSB of the imaginary part of the FPGA
60  ;          output will be read to bit 11 of x0, the LSB of the FPGA
61  ;          output will be read to bit 0 of x0.
62  ;          Writing data goes in the same bit order.
63  ;
64  ;-------------------------------------------------------------------------
65  ; Files:  pass.dat intequ.asm, ioequ.asm,
66  ;         vec_pass.asm. (Please list additional required program files.)
67  ;         flash.asm
68  ;-------------------------------------------------------------------------
69  ; Notes: Stack pointer initialization changed, SP set to some arbitrary
70  ;        address, future programmer must check this.
71  ;
72  ;
73  ;-------------------------------------------------------------------------
74  ; Address Registers:  (Please describe what each points to)
75  ;  r0 -> I/O address       r4 ->
76  ;  r1 ->                   r5 ->
77  ;  r2 ->                   r6 -> Reserved for Stack Pointer
78  ;  r3 ->                   r7 ->
79  ;
80  ;-------------------------------------------------------------------------
81  ;Memory Map: No memory used
82  ;
83  ; X Memory
84  ;------------
85  ; $00000a - $001BFF -- User internal (on-chip) memory
86  ; $001C00 - $00FFFF -- (Unavailable)
87  ; $010000 - $013FFF -- User external (off-chip) memory
88  ; STACK resides after last thing in external X memory
89  ;
90  ; Y Memory
91  ;------------
92  ; $000000 - $001BFF -- User internalmemory
93  ; $001C00 - $00FFFF -- (Unavailable)
94  ; $010000 - $013FFF -- User external (off-chip) memory
95  ;*************************************************************************
96  CODE_ST
97        nolist
98        include 'ioequ.asm'
99        include 'intequ.asm'
100       include 'vect.asm'
101       list
102
103 ;        org     x:0
104       include 'pass.dat'      ;include user data
105
106 ;        org     x:      ;locate stack after last thing in X memory in
107 PASS.DAT
108 ;STACK   equ     *       ;STACK is used in TXRX_ISR.ASM and must be allowed
109 to grow
110
111 STACK   equ     $200
112
113       org     p:$000100
114 START
```

```
;main_pr
      movep   #$040003,x:M_PCTL       ;set PLL for MPY of 4X
      movep   #$012422,x:M_BCR        ;set up one ext. wait state for all
AAR areas
      movep   #AAR0,x:M_AAR0          ;memory partition stuff (see p.4-6--4-
8 in
      movep   #AAR3,x:M_AAR3          ;DSP56302EVM UM)
      movep   #$012421,x:M_BCR        ;set up one ext. wait state for all
AAR areas
      ori     #3,mr                   ;mask interrupts
      movec   #0,sp                   ;clear hardware stack pointer
      move    #$004000,omr            ;operating mode 0 (b14 set for ext mem
access)
;        move    #STACK,r6               ;initialise stack pointer
      move    #STACK,r6               ;initialise stack pointer
      move    #-1,m6                  ;linear addressing of stack
;;;;;;;;;;;;;;;;;;from main until here the same as in
pass.asm;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;NEW;;;;NEW;;;;NEW;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;

;;;;;;;Disable interrupts previously used by the
CODEC;;;;;;;;;;;;;;;;;;;;;;;;
;M_IPRP = Memory mapped register___Interrupt Priority Register for
Peripherals
;UserManual 4-13
      move    x:M_IPRP,a1
      move    #$fffc00,x0
      and     x0,a                    ;Disable Host, ESSI, SCI and Timer
      nop
      move    a1,x:M_IPRP             ;interrupts


;;;;;;;Initialize IRQ
A;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;M_IPRC = Memory mapped register___Interrupt Priority Register for onChip
reg.
;UserManual 4-13
      move    x:M_IPRC,a1
      move    #$fffff8,x0             ;clear the bits corresponding to IRQA,
      and     x0,a                    ;leave all other bits unaffected
      nop
      move    #$000107,x0
      ;Bit 0 to 2 are for interrupt IRQ A, which is used. That's ok
      ;But bit 8 must be set too - otherwise no operation, reason
???????????
      or      x0,a                    ;Enable IRQ A, edge triggered, high
      nop                             ;priority
      move    a1,x:M_IPRC


;;;;;;;;Initialize AA2
pin;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;family manual 2-20
```

```
172         movep   #AAR2,x:M_AAR2
173
174
175
176
177
178   ;;;;;;;Set the # of wait states for AAR2 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
179   ;This wait states are inserted during accesses to memory areas not covered
180   ;by the AAR0....3 registers (FamilyManual 2-24)
181   ;M_BCR = Memory mapped register___Bus Control Register
182         move    x:M_BCR,a1
183         move    #$ffe3ff,X0
184         and     x0,a
185         ;move    #$000c00,X0    ;not enough wait states for wire wrap board,
186                                  ;may be enough for well designed PCB
187         move    #$001400,X0     ;this # of waitstates necessary
188         or      x0,a
189         nop
190         move    a1,x:M_BCR
191
192
193
194   ;;;;;;;;Enable all interrupts (clear the mask
195   flags);;;;;;;;;;;;;;;;;;;;;;;;;
196   ;mr is part of the Status register (FamilyManual 6-10)
197         andi    #$fc,mr                  ;enable all interrupts
198
199
200
201   ;;;;;;;;I/O address into r0;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
202         move #$18000,r0
203         nop
204         nop
205         nop
206
207
208
209   ;The code in this loop is written under the assumption, that "NEW_SA_FLAG"
210   ;is set, if an interrupt occurs.
211   ;Look vect.asm for more info.
212
213   process_loop
214         ;clr     a
215         ;move    x:NEW_SA_FLAG,a
216         ;cmp     #0,a              ;check if new sample available
217         ;jeq     NO_SAMP           ;if not, do nothing
218         ;move    x:(r0),x0
219         ;       .
220         ;       .
221         ;  processing
222         ;       .
223         ;       .
224         ;move     x0,x:(r0)
225   NO_SAMP
226         jmp process_loop
227
228
```

```
229
230
231
232   ;;;;;;;;Has to be used if interrupt service takes more than two commands,
233   ;;;;;;;;otherwise can use "fast interrupt" (code in the vector table)
234   ;Look vect.asm for more info
235   IRQ_A_service
236         move x:(r0),x0
237         ;processing
238         ;  nop
239         ;  nop
240         move x0,x:(r0)
241         rti
242
243   echo
244   CODE_END
245
246
247          include 'flash.asm'
248
249
250         end
251
252
253
254
255
256
257
258
```

```
1           page    132,66
2   ;***************************************************************************
3   ;   FLASH.ASM - routine to make user code bootable from a flash eeprom
4   ;
5   ;   1 June 1996 : version 1.0   rlr/jp
6   ;
7   ;   copyright (C) Motorola 1996
8   ;
9   ;   execute with:  R0 pointing to first word of user code
10  ;                  R1 containing size of the user's code in WORDS
11  ;***************************************************************************
12          nolist
13          ;include 'ioequ.asm'
14          list
15
16  AAR0V           EQU   $000909     ; value programmed into AAR0 (SRAM Enable)
17    ;  (32k words ext. SRAM located at p:$001000-$007FFF)
18    ; compare 9 most significant bits, look for a match with address
19    ; X:0000 0000 0xxx xxx xxxx xxxx, no packing, no muxing, P enabled,
20    ; X and Y disabled, AAR0 pin active low, asynchronous SRAM-type access
21
22  AAR1V           EQU   $010811     ; value programmed into AAR1 (FLASH Enable)
23    ;   (64k bytes FLASH located at x:$010000-$01FFFF)
24    ; compare 8 most significant bits, look for a match with address
25    ; X:0000 0001 xxxx xxx xxxx xxxx, no packing, no muxing, X enabled,
26    ; P and Y disabled, AAR1 pin active low, asynchronous SRAM-type access
27
28  BCRV            EQU   $012581     ; value programmed into BCR
29                                    ; (12 wait states for AAR1, 1 elsewhere)
30
31  ;***************************************************************************
32  .
33          org     p:$00800
34  flash
35  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
36          move    #CODE_ST,r0
37          move    #(CODE_END-CODE_ST),r1
38  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
39
40          movep   #$040001,x:M_PCTL  ; MF=2 (34 MHz)
41          movep   #AAR0V,x:M_AAR0    ; maps SRAM from p:$0000-$7fff
42          movep   #AAR1V,x:M_AAR1    ; set up flash space to be mapped from
43                                     ; x:$10000 to x:$1ffff (size 65536 bytes)
44          movep   #BCRV,x:M_BCR      ; 12 wait states in AAR1 domain,
45                                     ;  (360ns cycle time)
46                                     ;  1 wait state elsewhere
47
48  ;----- compute the number of sectors to write/read
49          move    #>42,x0
50          clr     b    r1,a
51
52  _cmpsec
53          inc     b
54          sub     x0,a
55          bgt     _cmpsec
56          move   b0,p:NUM_SEC
57
```

```
58  ;---------------------------------------------------------------------
59  ;  Initialize the loader code with the size of the user's code and
60  ;  the address of the user's code.   Then write the loader routine
61  ;  into the boot sector (the 1st 128-bytes) of the FLASH.
62  ;-----
63          move    r0,p:CODE_START     ; store the address of the user's code
64          move    r1,p:CODE_SIZE      ; store the size    of the user's code
65          move    #BOOT,r0            ; first DSP address to save
66          move    #$010000,r2         ; first address of FLASH
67          bsr     <WR_FLASH           ; put boot sector in flash
68  ;--- boot sector written
69
70
71  ;---------------------------------------------------------------------
72  ;  write the user's code into the FLASH.
73  ;-----
74          nop
75          move    #$10080,r2          ; user's code begins at flash address $80
76          move    p:CODE_START,r0     ; where to get the user's code
77          move    p:NUM_SEC,r3        ; the number of sectors to write
78
79          dor     r3,_wrsec
80          bsr     <WR_FLASH           ; WR_FLASH loads 42 words each time
81          nop
82  _wrsec
83
84  ;----- user code written
85
86          debug
87          bra     *
88
89
90  ;*********************************************************************
91  ;  this is the bootstrap loader
92  ;  it is loaded into the P:memory at boot time, sets up AAR0,
93  ;  then moves the user's code into 56303 pram beginning at p:0
94  ;
95  ;  THE LOADER WILL SET UP AAR0 TO MAKE THE 56303 MEMORY MAP AS FOLLOWS:
96  ;     (map not drawn proportionally to scale, but you get the point)
97  ;
98  ;                P:$FFFFFF  --------
99  ;                          | unused |
100 ;                          | ext P  |
101 ;                          | space  |
102 ;                          |        |
103 ;                P:$01FFFF |--------|
104 ;                          | FLASH  |
105 ;                P:$010000 |--------|
106 ;                          |        |
107 ;                          | unused |
108 ;                          | ext P  |
109 ;                          | space  |
110 ;                          |        |
111 ;                P:$007FFF |--------| <-- end of accessible P memory
112 ;                          |        |
113 ;                          |        |
114 ;                          |        |
```

```
115  ;                              | 28K    |
116  ;                              | ext P  |\
117  ;                              | (SRAM) | \
118  ;                              |        |  \  user's program will reside here,
119  ;                  P:$001000   |--------|  /  note there is no gap btw int and
120  ext P
121  ;                              | 4K int | /
122  ;                              | P space|/
123  ;                  P:$000000    --------
124  ;                                56303
125  ;                             P memory map
126  ;
127  ;*************************************************************************
128  BOOT
129          dc      BOOT_END-BOOT              ;storage for size of boot
130          dc      bootgo                     ;storage for address of boot
131  bootgo
132
133  ; set up AAR0 to create the memory map shown above
134          movep   #$040001,x:M_PCTL  ; MF=2 (34 MHz)
135          movep   #AAR0V,x:M_AAR0    ; maps SRAM from p:$0000-$7fff
136          movep   #AAR1V,x:M_AAR1    ; set up flash space to be mapped from
137                                     ; x:$10000 to x:$1ffff (size 65536 bytes)
138          movep   #BCRV,x:M_BCR      ; 31 wait states in AAR1 domain, 1 wait
139  state
140                                     ; elsewhere, THIS WILL CHANGE
141
142
143  ;-------------------------------------------------------------------------
144  -
145  ;The following code moves the user's code from the flash to P:RAM.
146  ;   The users code starts in the 2nd sector of the FLASH at P:$010080
147  ;-------------------------------------------------------------------------
148  -
149
150
151  load    move    p:NUM_SEC,r3              ; number of sectors to read
152  ;        move    p:CODE_START,r0           ; starting address of user code
153          move    #0,r0                     ; starting address of user code
154          move    p:CODE_SIZE,r2            ; R2 contains the length of code in
155  WORDS
156          move    #$010080,r1               ; user's code goes into FLASH at
157  $010080
158
159          dor     r3,_sector_loop           ; get all sectors
160          dor     #42,_word_loop            ; 42 words per FLASH sector
161          dor     #3,_byte_loop             ; each word requires 3 bytes
162          move    x:(r1)+,a2                ; Load byte from flash.
163          asr     #8,a,a                    ; Shift 8 times to move byte into
164                                            ; correct position.
165  _byte_loop
166          movem   a1,p:(r0)+                ; Stores 24-bit word into P mem
167          nop                               ; dummy NOP to avoid do loop
168  restrict.
169  _word_loop
170          move    (r1)+
171          move    (r1)+
```

```
_sector_loop

        move   p:CODE_START,r1
        jmp    (r1)                        ; Jump to the user's code

CODE_SIZE   ds   1                         ; user code size
CODE_START  ds   1                         ; starting location of user code
NUM_SEC     ds   1                         ; number of sectors to save

BOOT_END        equ     *


;=========================================================================
;*************************************************************************
;   WR_FLASH
;   This routine writes one 128-byte FLASH sector.
;      Each sector will contain 42 (24-bit) words from DSP RAM (126 bytes)
;      The last 2 bytes contain a 16-bit checksum of the data
;
;   enter with:  r2 pointing to starting PEROM Byte Address from 56303
p.o.v.
;                r0 pointing to first DSP word to save
;
;   corrupts:    r0 - points to next P:memory location to save
;                r2 - points to next FLASH byte address to write
;                a,b,x0
;-------------------------------------------------------------------------
--

WR_FLASH
        nop
        clr     b                          ;initialize checksum
;-------------------------------------------------------------------------
-
;  disable_protect - Software data Protection routine
;-------------------------------------------------------------------------
-
disable_protect
        move    #>$71,x0                   ; $71 ==> $AA at Flash
        move    x0,x:$15555                ;Atmel needs $5555
        move    #>$8E,x0                   ; $8E ==> $55 at Flash
        move    x0,x:$12AAA                ;Atmel needs $2AAA
        move    #>$41,x0                   ; $41 ==> $A0 at Flash
        move    x0,x:$15555                ;Atmel needs $5555
   ;--- writes are now enabled

        dor     #42,move_code              ;move 126 bytes
        move    p:(r0)+,a                  ;get 3 byte word
        add     a,b                        ;accumulate checksum
        move    a1,x:(r2)+                 ;move low byte to flash
        lsr     #8,a
        move    a1,x:(r2)+                 ;move mid byte
        lsr     #8,a
        move    a1,x:(r2)+                 ;move high byte
move_code

        ;*** 127th and 128th bytes are 16-bit checksum
```

```
229             move    b1,x:(r2)+              ;move low byte of checksum to flash
230             lsr     #8,b                    ;get next 8 bits of checksum
231             move    b1,x:(r2)+              ;move mid byte of checksum to flash
232
233             bsr    <delay_20m               ;wait 20 milliseconds to finish
234                                             ;...program cycle
235             rts
236
237     ;---------------------------------------------------------------------------
238     ; delay_20m --
239     ; 20.15 ms of delay guarantees that the write cycle has finished.
240     ;
241     ; For Write Pulse Width High to extend 150 us requires a minimum of
242     ; 5100 Icycles for a 33.868 MHz clock.  (29.5 ns/Icycle)
243     ;
244     ; the tWC is 20ms, so we just wait long enough here for the
245     ; write cycle to have started AND finished...
246     ;
247     ;---------------------------------------------------------------------------
248     delay_20m
249             dor     #110,_l1                ;110 x 207us = 22.7 ms
250             move    #7000,x0
251             rep     x0                      ; 207us delay
252             nop
253             nop
254     _l1
255             rts
256
257
258
259
260
261
```

```
1   ;
2           page    132,60
3   ;*****************************************************************************
4   *
5   ;   VECTORS.ASM
6   ;   Vector table for the 56303
7   ;
8   ;   Copyright (c) MOTOROLA 1996
9   ;             Semiconductor Products Sector
10  ;             Digital Signal Processing Division
11  ;
12  ;*****************************************************************************
13  *
14  ;
15            ORG       P:0
16  ;
17  vectors   JMP       START                          ; Hardware RESET
18  ;
19            jmp       *
20            NOP                          ;  Stack Error
21
22            jmp       *
23            NOP                          ;- Debug Request Interrupt
24  ;
25            jmp       *
26            NOP                          ;- Debug Request Interrupt
27  ;
28            jmp       *
29            NOP                          ;- Trap
30  ;
31            jmp       *
32            NOP                          ;- NMI
33  ;
34            NOP                          ;- Reserved
35            NOP
36  ;
37            NOP                          ;- Reserved
38            NOP
39
40  ;;;;;;;;;This is the fast version, applicable if no more than two commands
41  ;;;;;;;;;necessary
42            move x:(r0),x0               ;IRQ A
43            move x0,x:(r0)
44  ;;;;;;;;;If more processing required, one of the following two options
45            ;can be chosen:
46            ;
47            ;1st option: jump to an IRQ service routine, do all processing
48            ;            there.
49            ;jmp    IRQ_A_service
50            ;
51            ;
52            ;
53            ;2nd option: Set a "new sample flag", check this flag frequently
54            ;            in the main processing loop. If it is set, then read
55            ;            the new sample, write a processed sample and clear
56            ;            the "new sample flag". All processing is done in the
57            ;            main loop.
```

```
58              ;move    #1,x0
59              ;move    x0,x:NEW_SA_FLAG
60              ;
61              ;
62              ;The author thinks, that the 2nd option is better, because certain
63              ;actions in an interrupt routine lead to trouble (system crash)
64              ;according to the author's experience. For example, the author
65              ;experienced, that accessing the register a in an interrupt
66              ;routine leads to a system crash. The reason is unclear for the
67              ;author.
68              ;(author = Stephan Berner, developed parallel DSP HW)
69
70
71
72              jmp       *
73              NOP                           ;- IRQB
74    ;
75              jmp       *
76              NOP                           ;- IRQC
77    ;
78    ;          jsr       echo               ;- IRQD
79              jmp       *
80              nop
81    ;
82              jmp       *
83              NOP                           ;- DMA Channel 0
84    ;
85              jmp       *
86              NOP                           ;- DMA Channel 1
87    ;
88              jmp       *
89              NOP                           ;- DMA Channel 2
90    ;
91              jmp       *
92              NOP                           ;- DMA Channel 3
93    ;
94              jmp       *
95              NOP                           ;- DMA Channel 4
96    ;
97              jmp       *
98              NOP                           ;- DMA Channel 5
99    ;
100             jmp       *
101             NOP                           ;- Timer 0 Compare
102   ;
103             jmp       *
104             NOP                           ;- Timer 0 Overflow
105   ;
106             jmp       *
107             NOP                           ;- Timer 1 Compare
108   ;
109             jmp       *
110             NOP                           ;- Timer 1 Overflow
111   ;
112             jmp       *
113             NOP                           ;- Timer 2 Compare
114   ;
```

```
115             jmp         *
116             NOP                             ;- Timer 2 Overflow
117     ;
118
119             NOP
120             jmp         *
121
122             NOP
123             jmp         *
124
125             NOP
126             jmp         *
127
128             NOP
129             jmp         *
130
131             NOP
132             jmp         *
133
134             NOP
135             jmp         *
136
137
138
139
140     ;           jsr         ssi_rx_isr                      ;- ESSI0 Receive Data
141     ;
142     ;           jsr         ssi_rxe_isr                     ;- ESSI0 Receive Data w/
143     Exception Status
144     ;
145     ;           jsr         ssi_rxls_isr                    ;- ESSI0 Receive last slot
146     ;.
147     ;           jsr         ssi_tx_isr                      ;- ESSI0 Transmit Data
148     ;
149     ;           jsr         ssi_txe_isr                     ;- ESSI0 Transmit Data w/
150     Exception Status
151     ;
152     ;           jsr         ssi_txls_isr                    ;- ESSI0 Transmit last
153     slot
154     ;
155             NOP                                         ;- Reserved
156             NOP
157     ;
158             NOP                                         ;- Reserved
159             NOP
160     ;
161             jmp         *
162             NOP                             ;- ESSI1 Receive Data
163     ;
164             jmp         *
165             NOP                             ;- ESSI1 Receive Data w/ Exception
166     Status
167     ;
168             jmp         *
169             NOP                             ;- ESSI1 Receive last slot
170     ;
171             jmp         *
```

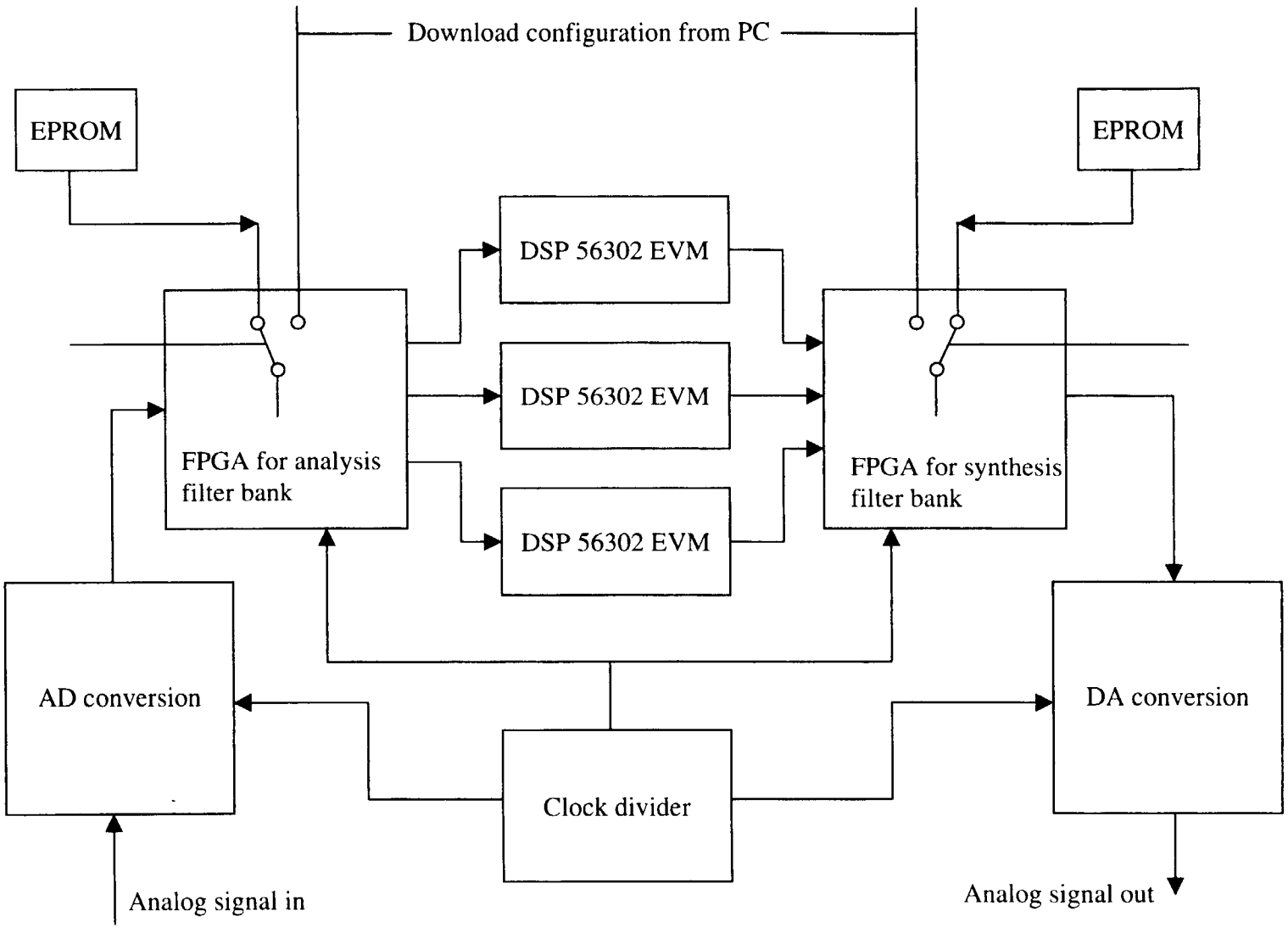
```
172             NOP                             ;- ESSI1 Transmit Data
173     ;
174             jmp         *
175             NOP                             ;- ESSI1 Transmit Data w/ Exception
176     Status
177     ;
178             jmp         *
179             NOP                             ;- ESSI1 Transmit last slot
180     ;
181
182             NOP                                         ;- Reserved
183             NOP
184     ;
185             NOP                                         ;- Reserved
186             NOP
187     ;
188             jmp         *
189             NOP                             ;- SCI Receive Data
190     ;
191             jmp         *
192             NOP                             ;- SCI Receive Data w/ Exception Status
193     ;
194             jmp         *
195             NOP                             ;- SCI Transmit Data
196     ;
197             jmp         *
198             NOP                             ;- SCI Idle Line
199     ;
200             jmp         *
201             NOP                             ;- SCI Timer
202     ;
203             NOP                                         ;- Reserved
204             NOP
205     ;
206             NOP                                         ;- Reserved
207             NOP
208     ;
209             NOP                                         ;- Reserved
210             NOP
211     ;
212     ;
213             jmp         *
214             NOP                             ; Host receive data full
215     ;
216     ;
217             jmp         *
218             NOP                             ;- Host transmit data empty.
219     ;
220             jmp         *
221             NOP                             ; Available for Host Command
222             jmp         *
223             NOP                             ; Available for Host Command
224             jmp         *
225             NOP                             ; Available for Host Command
226             jmp         *
227             NOP                             ; Available for Host Command
228             jmp         *
```

```
229             NOP                     ; Available for Host Command
230             jmp     *
231             NOP                     ; Available for Host Command
232             jmp     *
233             NOP                     ; Available for Host Command
234             jmp     *
235             NOP                     ; Available for Host Command
236             jmp     *
237             NOP                     ; Available for Host Command
238             jmp     *
239             NOP                     ; Available for Host Command
240             jmp     *
241             NOP                     ; Available for Host Command
242             jmp     *
243             NOP                     ; Available for Host Command
244             jmp     *
245             NOP                     ; Available for Host Command
246             jmp     *
247             NOP                     ; Available for Host Command
248             jmp     *
249             NOP                     ; Available for Host Command
250             jmp     *
251             NOP                     ; Available for Host Command
252             jmp     *
253             NOP                     ; Available for Host Command
254             jmp     *
255             NOP                     ; Available for Host Command
256             jmp     *
257             NOP                     ; Available for Host Command
258             jmp     *
259             NOP                     ; Available for Host Command
260             jmp     *
261             NOP                     ; Available for Host Command
262             jmp     *
263             NOP                     ; Available for Host Command
264             jmp     *
265             NOP                     ; Available for Host Command
266             jmp     *
267             NOP                     ; Available for Host Command
268             jmp     *
269             NOP                     ; Available for Host Command
270             jmp     *
271             NOP                     ; Available for Host Command
272             jmp     *
273             NOP                     ; Available for Host Command
274             jmp     *
275             NOP                     ; Available for Host Command
276             jmp     *
277             NOP                     ; Available for Host Command
278             jmp     *
279             NOP                     ; Available for Host Command
280             jmp     *
281
282              NOP                     ; Available for Host Command
283             jmp     *
284             NOP                     ; Available for Host Command
285             jmp     *
```

```
286             NOP                     ; Available for Host Command
287             jmp     *
288             NOP                     ; Available for Host Command
289             jmp     *
290             NOP                     ; Available for Host Command
291             jmp     *
292             NOP                     ; Available for Host Command
293             jmp     *
294             NOP                     ; Available for Host Command
295             jmp     *
296             NOP                     ; Available for Host Command
297             jmp     *
298             NOP                     ; Available for Host Command
299             jmp     *
300             NOP                     ; Available for Host Command
301             jmp     *
302             NOP                     ; Available for Host Command
303             jmp     *
304             NOP                     ; Available for Host Command
305             jmp     *
306             NOP                     ; Available for Host Command
307             jmp     *
308             NOP                     ; Available for Host Command
309             jmp     *
310             NOP                     ; Available for Host Command
311             jmp     *
312             NOP                     ; Available for Host Command
313             jmp     *
314             NOP                     ; Available for Host Command
315             jmp     *
316             NOP                     ; Available for Host Command
317             jmp     *
318             NOP                     ; Available for Host Command
319             jmp     *
320             NOP                     ; Available for Host Command
321             jmp     *
322             NOP                     ; Available for Host Command
323             jmp     *
324             NOP                     ; Available for Host Command
325             jmp     *
326             NOP                     ; Available for Host Command
327             jmp     *
328             NOP                     ; Available for Host Command
329             jmp     *
330             NOP                     ; Available for Host Command
331             jmp     *
332             NOP                     ; Available for Host Command
333             jmp     *
334             NOP                     ; Available for Host Command
335             jmp     *
336             NOP                     ; Available for Host Command
337             jmp     *
338             NOP                     ; Available for Host Command
339             jmp     *
340             NOP                     ; Available for Host Command
341             jmp     *
342             NOP                     ; Available for Host Command
```

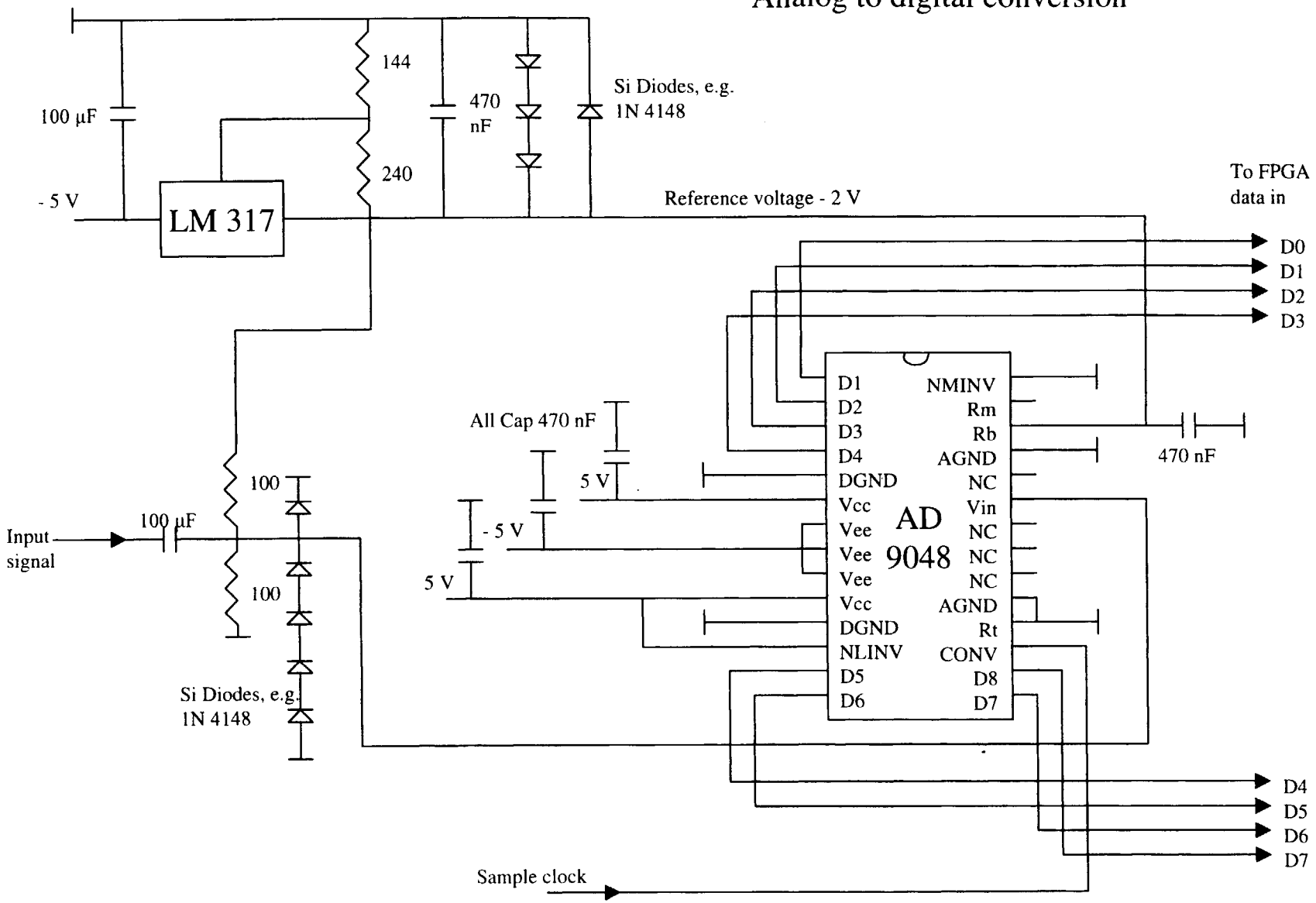
```
343             jmp     *
344
345
346              NOP                     ; Available for Host Command
347             jmp     *
348             NOP                     ; Available for Host Command
349             jmp     *
350             NOP                     ; Available for Host Command
351             jmp     *
352             NOP                     ; Available for Host Command
353             jmp     *
354             NOP                     ; Available for Host Command
355             jmp     *
356             NOP                     ; Available for Host Command
357             jmp     *
358             NOP                     ; Available for Host Command
359             jmp     *
360             NOP                     ; Available for Host Command
361             jmp     *
362             NOP                     ; Available for Host Command
363             jmp     *
364             NOP                     ; Available for Host Command
365             jmp     *
366             NOP                     ; Available for Host Command
367             jmp     *
368             NOP                     ; Available for Host Command
369             jmp     *
370             NOP                     ; Available for Host Command
371             jmp     *
372             NOP                     ; Available for Host Command
373             jmp     *
374             NOP                     ; Available for Host Command
375             jmp     *
376             NOP                     ; Available for Host Command
377             jmp     *
378             NOP                     ; Available for Host Command
379   ;
380   ;
```

Appendix F
Block diagram of hardware

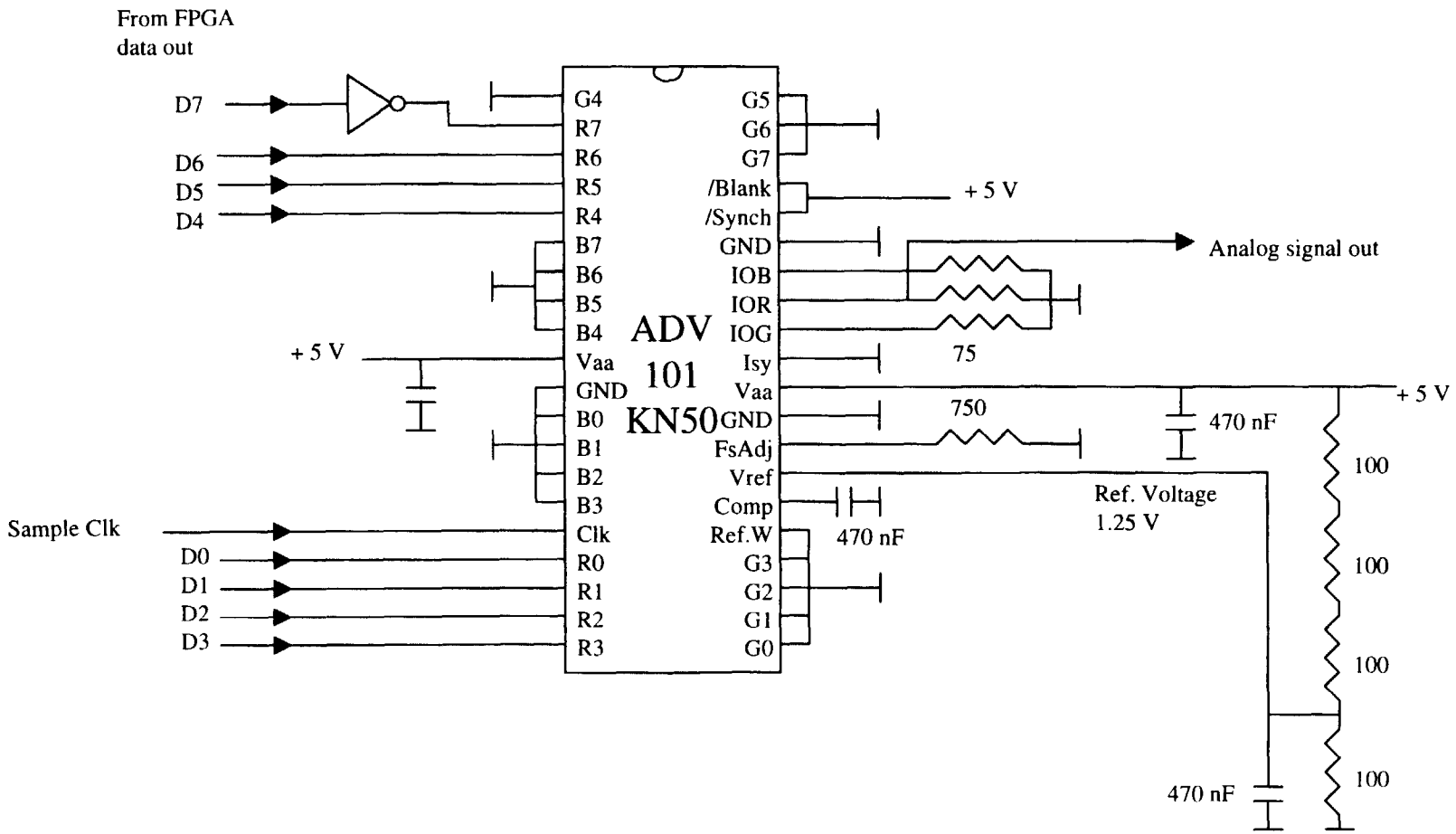

Download configuration from PC
EPROM
EPROM
DSP 56302 EVM
DSP 56302 EVM
DSP 56302 EVM
FPGA for analysis filter bank
FPGA for synthesis filter bank
AD conversion
DA conversion
Clock divider
Analog signal in
Analog signal out

Appendix G
Schematics

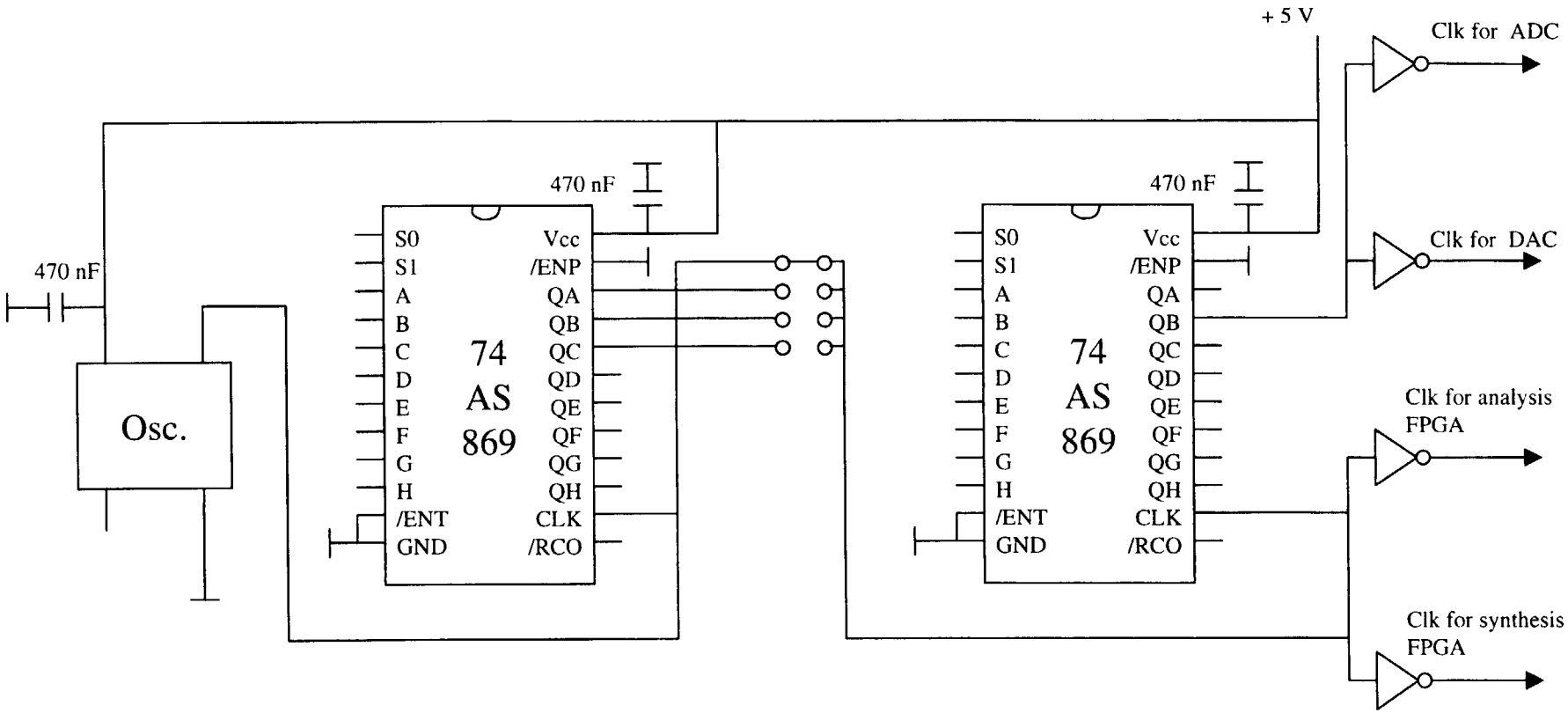
Analog to digital conversion



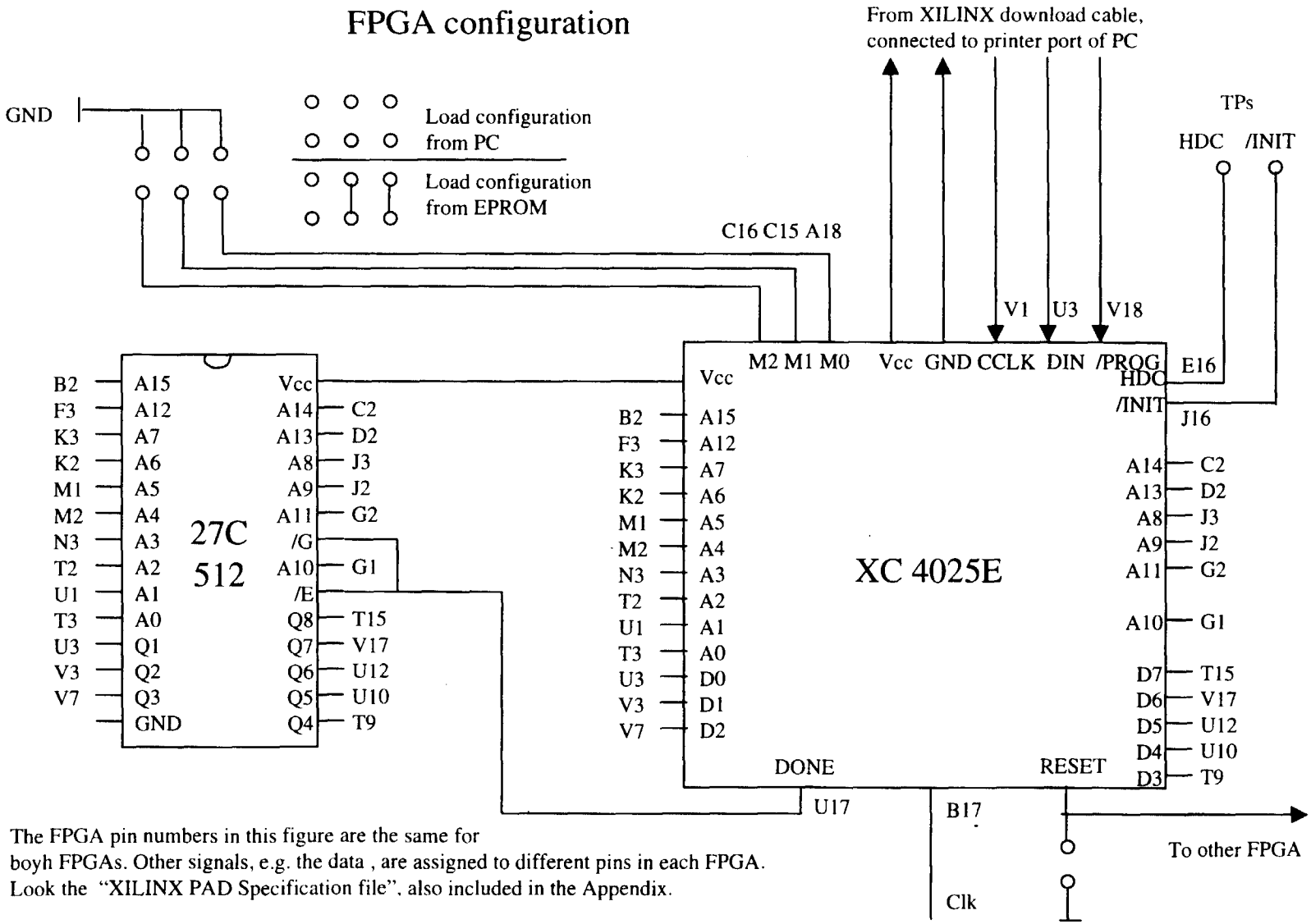
Digital to analog conversion



Clock divider



FPGA configuration

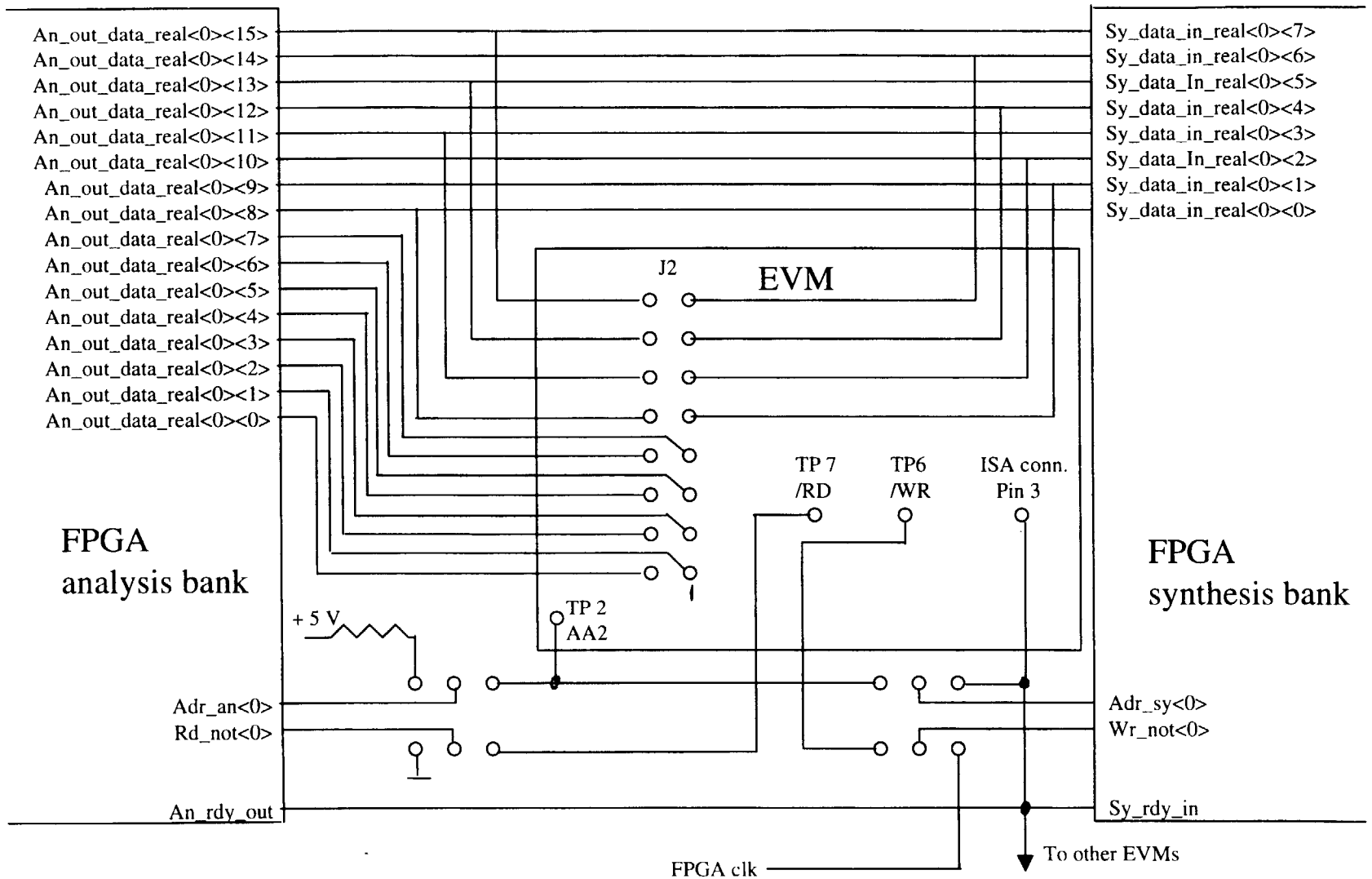


The FPGA pin numbers in this figure are the same for
boyh FPGAs. Other signals, e.g. the data , are assigned to different pins in each FPGA.
Look the "XILINX PAD Specification file", also included in the Appendix.

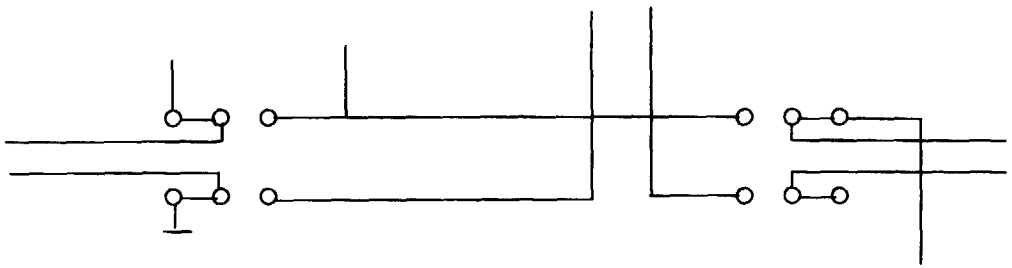
Burning the EPROM: 1) Take the *.bit file, generated by XILINX FNDTN, load address 45h until EOF into the EPROM at address 0

2) Very important: The bit order has to be reversed ! Can be done per software (flipping the bytes in the file), or per hardware (connecting bit 7 of EPROM to bit 0 of FPGA,....). This figure assumes, that the bytes are flipped in the file.

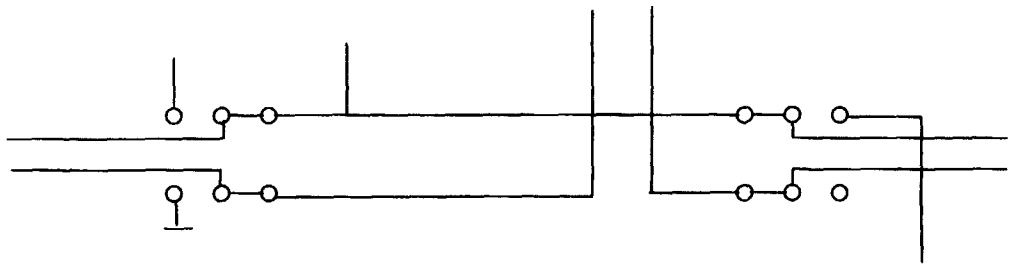
Interfacing DSP 56302 EVM to FPGAs



Jumper settings



Setting the jumpers to this position and removing the EVM allows pass through of the subband signal, no change of the wiring required.



Jumper setting for EVM operation

If the subband signal is complex, as in subband #2 (the bandpass), J1 of the EVM is also involved. Bit 15...bit 4 of the real part are connected to DDB23..... DDB12 of the EVM, bit 15....bit 4 of the imaginary part are connected to DDB11....DDB0 of the EVM.

Appendix H
FPGA pin assignments

```
R: Xilinx Place And Route M1.5.25.
pyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.
n Jun 28 17:55:51 1999

linx PAD Specification File
**************************

put file:       map.ncd
tput file:      an4.ncd
rt type:        xc4025e
eed grade:      -3
ckage:          pg223

n Jun 28 17:55:51 1999

nout by Pin Name:
```

Pin Name	Direction	Pin Number
Adr_An<0>	INPUT	G15
Adr_An<1>	INPUT	H15
Adr_An<2>	INPUT	J17
An_data_in<0>	INPUT	E1
An_data_in<1>	INPUT	H3
An_data_in<2>	INPUT	C1
An_data_in<3>	INPUT	H4
An_data_in<4>	INPUT	D1
An_data_in<5>	INPUT	G4
An_data_in<6>	INPUT	F2
An_data_in<7>	INPUT	F1
An_data_out_imag<0><0>	OUTPUT	P4
An_data_out_imag<0><10>	OUTPUT	U6
An_data_out_imag<0><11>	OUTPUT	U5
An_data_out_imag<0><12>	OUTPUT	T6
An_data_out_imag<0><13>	OUTPUT	R2
An_data_out_imag<0><14>	OUTPUT	P3
An_data_out_imag<0><15>	OUTPUT	N4
An_data_out_imag<0><1>	OUTPUT	U11
An_data_out_imag<0><2>	OUTPUT	R12
An_data_out_imag<0><3>	OUTPUT	M18
An_data_out_imag<0><4>	OUTPUT	R18
An_data_out_imag<0><5>	OUTPUT	T18
An_data_out_imag<0><6>	OUTPUT	N16
An_data_out_imag<0><7>	OUTPUT	T11
An_data_out_imag<0><8>	OUTPUT	R11
An_data_out_imag<0><9>	OUTPUT	V6
An_data_out_imag<1><0>	OUTPUT	L15
An_data_out_imag<1><10>	OUTPUT	V9
An_data_out_imag<1><11>	OUTPUT	U7
An_data_out_imag<1><12>	OUTPUT	R5
An_data_out_imag<1><13>	OUTPUT	U8
An_data_out_imag<1><14>	OUTPUT	T8
An_data_out_imag<1><15>	OUTPUT	L18
An_data_out_imag<1><1>	OUTPUT	L16
An_data_out_imag<1><2>	OUTPUT	N15
An_data_out_imag<1><3>	OUTPUT	P15
An_data_out_imag<1><4>	OUTPUT	N17
An_data_out_imag<1><5>	OUTPUT	R17
An_data_out_imag<1><6>	OUTPUT	U16
An_data_out_imag<1><7>	OUTPUT	U18
An_data_out_imag<1><8>	OUTPUT	R14

An_data_out_real<2><6>	OUTPUT	H17
An_data_out_real<2><7>	OUTPUT	C17
An_data_out_real<2><8>	OUTPUT	A15
An_data_out_real<2><9>	OUTPUT	C9
An_rdy_out	OUTPUT	P18
Rd_not<0>	INPUT	P1
Rd_not<1>	INPUT	K18
Rd_not<2>	INPUT	H18
clk	INPUT	B17
reset	INPUT	E2

Dedicated or Special Pin Name	Pin Number
/PROG	V18
CCLK	V1
DONE	U17
GND	R9
GND	T12
GND	D9
GND	M16
GND	R3
GND	K15
GND	T7
GND	D4
GND	R16
GND	K4
GND	M3
GND	G16
GND	D15
GND	C12
GND	G3
GND	C7
M0	A18
M1	C15
M2	C16
TDO	U2
TDO	TDO
VCC	R15
VCC	D10
VCC	J4
VCC	J15
VCC	R10
VCC	R4
VCC	D3
VCC	D16

Pinout by Pin Number:

Pin Number	Pin Name	Direction	Constraint
A2			PROHIBITED
A3	An_data_out_real<2><11>	OUTPUT	LOCATED
A4	An_data_out_real<1><9>	OUTPUT	LOCATED
A5			
A6	An_data_out_real<0><13>	OUTPUT	LOCATED
A7	An_data_out_real<1><13>	OUTPUT	LOCATED
A8	An_data_out_real<2><10>	OUTPUT	LOCATED
A9			
A10	An_data_out_real<1><8>	OUTPUT	LOCATED
A11	An_data_out_real<1><7>	OUTPUT	LOCATED

A12			
A13			
A14	An_data_out_real<0><8>	OUTPUT	LOCATED
A15	An_data_out_real<2><8>	OUTPUT	LOCATED
A16			
A17	An_data_out_real<0><4>	OUTPUT	LOCATED
A18	M0		
B1			
B2			
B3			
B4			PROHIBITED
B5	An_data_out_real<2><12>	OUTPUT	LOCATED
B6	An_data_out_real<1><12>	OUTPUT	LOCATED
B7			PROHIBITED
B8	An_data_out_real<2><13>	OUTPUT	LOCATED
B9			
B10	An_data_out_real<0><14>	OUTPUT	LOCATED
B11			
B12			
B13	An_data_out_real<0><15>	OUTPUT	LOCATED
B14	An_data_out_real<0><7>	OUTPUT	LOCATED
B15			
B16	An_data_out_real<0><3>	OUTPUT	LOCATED
B17	clk	INPUT	LOCATED
B18	An_data_out_real<0><5>	OUTPUT	LOCATED
C1	An_data_in<2>	INPUT	LOCATED
C2			PROHIBITED
C3			PROHIBITED
C4			PROHIBITED
C5			
C6	An_data_out_real<2><15>	OUTPUT	LOCATED
C7	GND		
C8	An_data_out_real<1><10>	OUTPUT	LOCATED
C9	An_data_out_real<2><9>	OUTPUT	LOCATED
C10			
C11	An_data_out_real<0><12>	OUTPUT	LOCATED
C12	GND		
C13			
C14			
C15	M1		
C16	M2		
C17	An_data_out_real<2><7>	OUTPUT	LOCATED
C18	An_data_out_real<1><2>	OUTPUT	LOCATED
D1	An_data_in<4>	INPUT	LOCATED
D2			PROHIBITED
D3	VCC		
D4	GND		
D5	An_data_out_real<1><11>	OUTPUT	LOCATED
D6	An_data_out_real<1><15>	OUTPUT	LOCATED
D7	An_data_out_real<1><14>	OUTPUT	LOCATED .
D8	An_data_out_real<2><14>	OUTPUT	LOCATED
D9	GND		
D10	VCC		
D11			
D12	An_data_out_real<0><11>	OUTPUT	LOCATED
D13	An_data_out_real<0><10>	OUTPUT	LOCATED
D14	An_data_out_real<0><9>	OUTPUT	LOCATED
D15	GND		
D16	VCC		
D17	An_data_out_real<0><6>	OUTPUT	LOCATED
D18	An_data_out_real<2><1>	OUTPUT	LOCATED

E1	An_data_in<0>	INPUT	LOCATED
E2	reset	INPUT	LOCATED
E3			
E4	An_data_out_real<0><1>	OUTPUT	LOCATED
E15	An_data_out_real<1><5>	OUTPUT	LOCATED
E16			PROHIBITED
E17			PROHIBITED
E18	An_data_out_real<1><6>	OUTPUT	LOCATED
F1	An_data_in<7>	INPUT	LOCATED
F2	An_data_in<6>	INPUT	LOCATED
F3			PROHIBITED
F4	An_data_out_real<0><2>	OUTPUT	LOCATED
F15	An_data_out_real<2><2>	OUTPUT	LOCATED
F16	An_data_out_real<1><1>	OUTPUT	LOCATED
F17	An_data_out_real<1><3>	OUTPUT	LOCATED
F18	An_data_out_real<2><3>	OUTPUT	LOCATED
G1			PROHIBITED
G2			PROHIBITED
G3	GND		
G4	An_data_in<5>	INPUT	LOCATED
G15	Adr_An<0>	INPUT	LOCATED
G16	GND		
G17	An_data_out_real<2><5>	OUTPUT	LOCATED
G18	An_data_out_real<2><4>	OUTPUT	LOCATED
H1			PROHIBITED
H2			
H3	An_data_in<1>	INPUT	LOCATED
H4	An_data_in<3>	INPUT	LOCATED
H15	Adr_An<1>	INPUT	LOCATED
H16	An_data_out_real<1><4>	OUTPUT	LOCATED
H17	An_data_out_real<2><6>	OUTPUT	LOCATED
H18	Rd_not<2>	INPUT	LOCATED
J1			PROHIBITED
J2			PROHIBITED
J3			PROHIBITED
J4	VCC		
J15	VCC		
J16			PROHIBITED
J17	Adr_An<2>	INPUT	LOCATED
J18	An_data_out_real<1><0>	OUTPUT	LOCATED
K1			PROHIBITED
K2			PROHIBITED
K3			PROHIBITED
K4	GND		
K15	GND		
K16	An_data_out_real<2><0>	OUTPUT	LOCATED
K17	An_data_out_imag<2><3>	OUTPUT	LOCATED
K18	Rd_not<1>	INPUT	LOCATED
L1			PROHIBITED
L2			
L3			
L4			
L15	An_data_out_imag<1><0>	OUTPUT	LOCATED
L16	An_data_out_imag<1><1>	OUTPUT	LOCATED
L17	An_data_out_imag<2><1>	OUTPUT	LOCATED
L18	An_data_out_imag<1><15>	OUTPUT	LOCATED
M1			PROHIBITED
M2			PROHIBITED
M3	GND		
M4	An_data_out_real<0><0>	OUTPUT	LOCATED
M15	An_data_out_imag<2><2>	OUTPUT	LOCATED

116	GND		
117	An_data_out_imag<2><15>	OUTPUT	LOCATED
118	An_data_out_imag<0><3>	OUTPUT	LOCATED
11			
12			
13			PROHIBITED
14	An_data_out_imag<0><15>	OUTPUT	LOCATED
115	An_data_out_imag<1><2>	OUTPUT	LOCATED
116	An_data_out_imag<0><6>	OUTPUT	LOCATED
117	An_data_out_imag<1><4>	OUTPUT	LOCATED
118	An_data_out_imag<2><0>	OUTPUT	LOCATED
21	Rd_not<0>	INPUT	LOCATED
22	An_data_out_imag<2><13>	OUTPUT	LOCATED
23	An_data_out_imag<0><14>	OUTPUT	LOCATED
24	An_data_out_imag<0><0>	OUTPUT	LOCATED
215	An_data_out_imag<1><3>	OUTPUT	LOCATED
216	An_data_out_imag<2><7>	OUTPUT	LOCATED
217			
218	An_rdy_out	OUTPUT	LOCATED
31			
32	An_data_out_imag<0><13>	OUTPUT	LOCATED
33	GND		
34	VCC		
35	An_data_out_imag<1><12>	OUTPUT	LOCATED
36	An_data_out_imag<2><14>	OUTPUT	LOCATED
37	An_data_out_imag<2><12>	OUTPUT	LOCATED
38	An_data_out_imag<2><11>	OUTPUT	LOCATED
39	GND		
310	VCC		
311	An_data_out_imag<0><8>	OUTPUT	LOCATED
312	An_data_out_imag<0><2>	OUTPUT	LOCATED
313			
314	An_data_out_imag<1><8>	OUTPUT	LOCATED
315	VCC		
316	GND		
317	An_data_out_imag<1><5>	OUTPUT	LOCATED
318	An_data_out_imag<0><4>	OUTPUT	LOCATED
T1			
T2			PROHIBITED
T3			PROHIBITED
T4			PROHIBITED
T5			
T6	An_data_out_imag<0><12>	OUTPUT	LOCATED
T7	GND		
T8	An_data_out_imag<1><14>	OUTPUT	LOCATED
T9			PROHIBITED
T10			
T11	An_data_out_imag<0><7>	OUTPUT	LOCATED
T12	GND		
T13			
T14	An_data_out_imag<2><6>	OUTPUT	LOCATED
T15			PROHIBITED
T16	An_data_out_imag<2><5>	OUTPUT	LOCATED
T17	An_data_out_imag<2><4>	OUTPUT	LOCATED
T18	An_data_out_imag<0><5>	OUTPUT	LOCATED
TDO	TDO		
U1			PROHIBITED
U2	TDO		
U3			PROHIBITED
U4			
U5	An_data_out_imag<0><11>	OUTPUT	LOCATED

U6	An_data_out_imag<0><10>	OUTPUT	LOCATED
U7	An_data_out_imag<1><11>	OUTPUT	LOCATED
U8	An_data_out_imag<1><13>	OUTPUT	LOCATED
U9			PROHIBITED
U10			PROHIBITED
U11	An_data_out_imag<0><1>	OUTPUT	LOCATED
U12			PROHIBITED
U13			
U14	An_data_out_imag<2><9>	OUTPUT	LOCATED
U15	An_data_out_imag<2><8>	OUTPUT	LOCATED
U16	An_data_out_imag<1><6>	OUTPUT	LOCATED
U17	DONE		
U18	An_data_out_imag<1><7>	OUTPUT	LOCATED
V1	CCLK		
V2			PROHIBITED
V3			PROHIBITED
V4			
V5			
V6	An_data_out_imag<0><9>	OUTPUT	LOCATED
V7			PROHIBITED
V8	An_data_out_imag<1><9>	OUTPUT	LOCATED
V9	An_data_out_imag<1><10>	OUTPUT	LOCATED
V10			
V11			
V12			PROHIBITED
V13			
V14	An_data_out_imag<2><10>	OUTPUT	LOCATED
V15			
V16			
V17			PROHIBITED
V18	/PROG		

```
#
# Pinout constraints listing
# These constraints are in PCF grammar format
# and may be cut and pasted into the PCF file
# after the "SCHEMATIC END ;" statement to
# preserve this pinout for future design iterations.
#
COMP "Adr_An<0>" LOCATE = SITE "G15" ;
COMP "Adr_An<1>" LOCATE = SITE "H15" ;
COMP "Adr_An<2>" LOCATE = SITE "J17" ;
COMP "An_data_in<0>" LOCATE = SITE "E1" ;
COMP "An_data_in<1>" LOCATE = SITE "H3" ;
COMP "An_data_in<2>" LOCATE = SITE "C1" ;
COMP "An_data_in<3>" LOCATE = SITE "H4" ;
COMP "An_data_in<4>" LOCATE = SITE "D1" ;
COMP "An_data_in<5>" LOCATE = SITE "G4" ;
COMP "An_data_in<6>" LOCATE = SITE "F2" ;
COMP "An_data_in<7>" LOCATE = SITE "F1" ;
COMP "An_data_out_imag<0><0>" LOCATE = SITE "P4" ;
COMP "An_data_out_imag<0><10>" LOCATE = SITE "[illegible]" ;
COMP "An_data_out_imag<0><11>" LOCATE = SITE "U5" ;
COMP "An_data_out_imag<0><12>" LOCATE = SITE "T6" ;
COMP "An_data_out_imag<0><13>" LOCATE = SITE "R2" ;
COMP "An_data_out_imag<0><14>" LOCATE = SITE "P3" ;
COMP "An_data_out_imag<0><15>" LOCATE = SITE "N4" ;
COMP "An_data_out_imag<0><1>" LOCATE = SITE "U11" ;
COMP "An_data_out_imag<0><2>" LOCATE = SITE "R12" ;
COMP "An_data_out_imag<0><3>" LOCATE = SITE "M18" ;
```

```
:P "An_data_out_imag<0><4>" LOCATE = SITE "R18" ;
(P "An_data_out_imag<0><5>" LOCATE = SITE "T18" ;
:P "An_data_out_imag<0><6>" LOCATE = SITE "N16" ;
(P "An_data_out_imag<0><7>" LOCATE = SITE "T11" ;
(P "An_data_out_imag<0><8>" LOCATE = SITE "R11" ;
(P "An_data_out_imag<0><9>" LOCATE = SITE "V6" ;
:P "An_data_out_imag<1><0>" LOCATE = SITE "L15" ;
:P "An_data_out_imag<1><10>" LOCATE = SITE "V9" ;
(P "An_data_out_imag<1><11>" LOCATE = SITE "U7" ;
:P "An_data_out_imag<1><12>" LOCATE = SITE "R5" ;
(P "An_data_out_imag<1><13>" LOCATE = SITE "U8" ;
:P "An_data_out_imag<1><14>" LOCATE = SITE "T8" ;
(P "An_data_out_imag<1><15>" LOCATE = SITE "L18" ;
(P "An_data_out_imag<1><1>" LOCATE = SITE "L16" ;
:P "An_data_out_imag<1><2>" LOCATE = SITE "N15" ;
:P "An_data_out_imag<1><3>" LOCATE = SITE "P15" ;
(P "An_data_out_imag<1><4>" LOCATE = SITE "N17" ;
(P "An_data_out_imag<1><5>" LOCATE = SITE "R17" ;
(P "An_data_out_imag<1><6>" LOCATE = SITE "U16" ;
(P "An_data_out_imag<1><7>" LOCATE = SITE "U18" ;
(P "An_data_out_imag<1><8>" LOCATE = SITE "R14" ;
(P "An_data_out_imag<1><9>" LOCATE = SITE "V8" ;
(P "An_data_out_imag<2><0>" LOCATE = SITE "N18" ;
(P "An_data_out_imag<2><10>" LOCATE = SITE "V14" ;
(P "An_data_out_imag<2><11>" LOCATE = SITE "R8" ;
(P "An_data_out_imag<2><12>" LOCATE = SITE "R7" ;
(P "An_data_out_imag<2><13>" LOCATE = SITE "P2" ;
(P "An_data_out_imag<2><14>" LOCATE = SITE "R6" ;
(P "An_data_out_imag<2><15>" LOCATE = SITE "M17" ;
(P "An_data_out_imag<2><1>" LOCATE = SITE "L17" ;
(P "An_data_out_imag<2><2>" LOCATE = SITE "M15" ;
(P "An_data_out_imag<2><3>" LOCATE = SITE "K17" ;
(P "An_data_out_imag<2><4>" LOCATE = SITE "T17" ;
(P "An_data_out_imag<2><5>" LOCATE = SITE "T16" ;
(P "An_data_out_imag<2><6>" LOCATE = SITE "T14" ;
(P "An_data_out_imag<2><7>" LOCATE = SITE "P16" ;
(P "An_data_out_imag<2><8>" LOCATE = SITE "U15" ;
(P "An_data_out_imag<2><9>" LOCATE = SITE "U14" ;
(P "An_data_out_real<0><0>" LOCATE = SITE "M4" ;
(P "An_data_out_real<0><10>" LOCATE = SITE "D13" ;
(P "An_data_out_real<0><11>" LOCATE = SITE "D12" ;
(P "An_data_out_real<0><12>" LOCATE = SITE "C11" ;
(P "An_data_out_real<0><13>" LOCATE = SITE "A6" ;
(P "An_data_out_real<0><14>" LOCATE = SITE "B10" ;
(P "An_data_out_real<0><15>" LOCATE = SITE "B13" ;
(P "An_data_out_real<0><1>" LOCATE = SITE "E4" ;
(P "An_data_out_real<0><2>" LOCATE = SITE "F4" ;
(P "An_data_out_real<0><3>" LOCATE = SITE "B16" ;
(P "An_data_out_real<0><4>" LOCATE = SITE "A17" ;
(P "An_data_out_real<0><5>" LOCATE = SITE "B18" ;
(P "An_data_out_real<0><6>" LOCATE = SITE "D17" ;
(P "An_data_out_real<0><7>" LOCATE = SITE "B14" ;
(P "An_data_out_real<0><8>" LOCATE = SITE "A14" ;
(P "An_data_out_real<0><9>" LOCATE = SITE "D14" ;
(P "An_data_out_real<1><0>" LOCATE = SITE "J18" ;
(P "An_data_out_real<1><10>" LOCATE = SITE "C8" ;
(P "An_data_out_real<1><11>" LOCATE = SITE "D5" ;
(P "An_data_out_real<1><12>" LOCATE = SITE "B6" ;
(P "An_data_out_real<1><13>" LOCATE = SITE "A7" ;
(P "An_data_out_real<1><14>" LOCATE = SITE "D7" ;
(P "An_data_out_real<1><15>" LOCATE = SITE "D6" ;
```

```
COMP "An_data_out_real<1><1>" LOCATE = SITE "F16" ;
COMP "An_data_out_real<1><2>" LOCATE = SITE "C18" ;
COMP "An_data_out_real<1><3>" LOCATE = SITE "F17" ;
COMP "An_data_out_real<1><4>" LOCATE = SITE "H16" ;
COMP "An_data_out_real<1><5>" LOCATE = SITE "E15" ;
COMP "An_data_out_real<1><6>" LOCATE = SITE "E18" ;
COMP "An_data_out_real<1><7>" LOCATE = SITE "A11" ;
COMP "An_data_out_real<1><8>" LOCATE = SITE "A10" ;
COMP "An_data_out_real<1><9>" LOCATE = SITE "A4" ;
COMP "An_data_out_real<2><0>" LOCATE = SITE "K16" ;
COMP "An_data_out_real<2><10>" LOCATE = SITE "A8" ;
COMP "An_data_out_real<2><11>" LOCATE = SITE "A3" ;
COMP "An_data_out_real<2><12>" LOCATE = SITE "B5" ;
COMP "An_data_out_real<2><13>" LOCATE = SITE "B8" ;
COMP "An_data_out_real<2><14>" LOCATE = SITE "D8" ;
COMP "An_data_out_real<2><15>" LOCATE = SITE "C6" ;
COMP "An_data_out_real<2><1>" LOCATE = SITE "D18" ;
COMP "An_data_out_real<2><2>" LOCATE = SITE "F15" ;
COMP "An_data_out_real<2><3>" LOCATE = SITE "F18" ;
COMP "An_data_out_real<2><4>" LOCATE = SITE "G18" ;
COMP "An_data_out_real<2><5>" LOCATE = SITE "G17" ;
COMP "An_data_out_real<2><6>" LOCATE = SITE "H17" ;
COMP "An_data_out_real<2><7>" LOCATE = SITE "C17" ;
COMP "An_data_out_real<2><8>" LOCATE = SITE "A15" ;
COMP "An_data_out_real<2><9>" LOCATE = SITE "C9" ;
COMP "An_rdy_out" LOCATE = SITE "P18" ;
COMP "Rd_not<0>" LOCATE = SITE "P1" ;
COMP "Rd_not<1>" LOCATE = SITE "K18" ;
COMP "Rd_not<2>" LOCATE = SITE "H18" ;
COMP "clk" LOCATE = SITE "B17" ;
COMP "reset" LOCATE = SITE "E2" ;
#
```

```
R: Xilinx Place And Route M1.5.25.
pyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.
d Jun 02 21:47:24 1999

linx PAD Specification File
**************************

put file:       map.ncd
tput file:      sy4.ncd
rt type:        xc4025e
eed grade:      -3
ckage:          pg223

d Jun 02 21:47:24 1999

nout by Pin Name:
```

Pin Name	Direction	Pin Number
Adr_Sy<0>	INPUT	T16
Adr_Sy<1>	INPUT	T5
Adr_Sy<2>	INPUT	C17
Sy_data_in_imag<0><0>	INPUT	M15
Sy_data_in_imag<0><1>	INPUT	M17
Sy_data_in_imag<0><2>	INPUT	P18
Sy_data_in_imag<0><3>	INPUT	R18
Sy_data_in_imag<0><4>	INPUT	N18
Sy_data_in_imag<0><5>	INPUT	P15
Sy_data_in_imag<0><6>	INPUT	N15
Sy_data_in_imag<0><7>	INPUT	N17
Sy_data_in_imag<1><0>	INPUT	U16
Sy_data_in_imag<1><1>	INPUT	R17
Sy_data_in_imag<1><2>	INPUT	T17
Sy_data_in_imag<1><3>	INPUT	N16
Sy_data_in_imag<1><4>	INPUT	T14
Sy_data_in_imag<1><5>	INPUT	T18
Sy_data_in_imag<1><6>	INPUT	U15
Sy_data_in_imag<1><7>	INPUT	P17
Sy_data_in_imag<2><0>	INPUT	V16
Sy_data_in_imag<2><1>	INPUT	P16
Sy_data_in_imag<2><2>	INPUT	R13
Sy_data_in_imag<2><3>	INPUT	R14
Sy_data_in_imag<2><4>	INPUT	T13
Sy_data_in_imag<2><5>	INPUT	U14
Sy_data_in_imag<2><6>	INPUT	V15
Sy_data_in_imag<2><7>	INPUT	M18
Sy_data_in_real<0><0>	INPUT	E3
Sy_data_in_real<0><1>	INPUT	B1
Sy_data_in_real<0><2>	INPUT	D8
Sy_data_in_real<0><3>	INPUT	C8
Sy_data_in_real<0><4>	INPUT	B6
Sy_data_in_real<0><5>	INPUT	D6
Sy_data_in_real<0><6>	INPUT	B12
Sy_data_in_real<0><7>	INPUT	D11
Sy_data_in_real<1><0>	INPUT	C5
Sy_data_in_real<1><1>	INPUT	B3
Sy_data_in_real<1><2>	INPUT	C6
Sy_data_in_real<1><3>	INPUT	A5
Sy_data_in_real<1><4>	INPUT	B5
Sy_data_in_real<1><5>	INPUT	D5
Sy_data_in_real<1><6>	INPUT	A12

Sy_data_in_real<1><7>	INPUT	D12
Sy_data_in_real<2><0>	INPUT	B11
Sy_data_in_real<2><1>	INPUT	A8
Sy_data_in_real<2><2>	INPUT	B9
Sy_data_in_real<2><3>	INPUT	A7
Sy_data_in_real<2><4>	INPUT	C11
Sy_data_in_real<2><5>	INPUT	C10
Sy_data_in_real<2><6>	INPUT	A3
Sy_data_in_real<2><7>	INPUT	A4
Sy_data_out<0>	OUTPUT	H17
Sy_data_out<10>	OUTPUT	R8
Sy_data_out<11>	OUTPUT	V9
Sy_data_out<12>	OUTPUT	U11
Sy_data_out<13>	OUTPUT	V8
Sy_data_out<14>	OUTPUT	L15
Sy_data_out<15>	OUTPUT	L18
Sy_data_out<16>	OUTPUT	J18
Sy_data_out<17>	OUTPUT	K16
Sy_data_out<18>	OUTPUT	U7
Sy_data_out<19>	OUTPUT	V6
Sy_data_out<1>	OUTPUT	G15
Sy_data_out<20>	OUTPUT	U6
Sy_data_out<21>	OUTPUT	R7
Sy_data_out<2>	OUTPUT	J17
Sy_data_out<3>	OUTPUT	H15
Sy_data_out<4>	OUTPUT	H18
Sy_data_out<5>	OUTPUT	L17
Sy_data_out<6>	OUTPUT	L16
Sy_data_out<7>	OUTPUT	T8
Sy_data_out<8>	OUTPUT	U8
Sy_data_out<9>	OUTPUT	V11
Sy_rdy_in	INPUT	U4
Wr_not<0>	INPUT	U18
Wr_not<1>	INPUT	P3
Wr_not<2>	INPUT	C14
clk	INPUT	B17
downsampl_del	INPUT	K18
reset	INPUT	K17

Dedicated or Special Pin Name	Pin Number
/PROG	V18
CCLK	V1
DONE	U17
GND	T12
GND	R3
GND	M3
GND	R16
GND	K15
GND	G3
GND	G16
GND	M16
GND	D15
GND	K4
GND	D9
GND	C12
GND	T7
GND	C7
GND	R9
GND	D4
M0	A18

M1	C15
M2	C16
TDO	TDO
TDO	U2
VCC	D16
VCC	D3
VCC	J4
VCC	D10
VCC	R10
VCC	R15
VCC	J15
VCC	R4

nout by Pin Number:

Pin Number	Pin Name	Direction	Constraint
A2			PROHIBITED
A3	Sy_data_in_real<2><6>	INPUT	LOCATED
A4	Sy_data_in_real<2><7>	INPUT	LOCATED
A5	Sy_data_in_real<1><3>	INPUT	LOCATED
A6			
A7	Sy_data_in_real<2><3>	INPUT	LOCATED
A8	Sy_data_in_real<2><1>	INPUT	LOCATED
A9			
A10			
A11			
A12	Sy_data_in_real<1><6>	INPUT	LOCATED
A13			
A14			
A15			
A16			
A17			
A18	M0		
B1	Sy_data_in_real<0><1>	INPUT	LOCATED
B2			
B3	Sy_data_in_real<1><1>	INPUT	LOCATED
B4			PROHIBITED
B5	Sy_data_in_real<1><4>	INPUT	LOCATED
B6	Sy_data_in_real<0><4>	INPUT	LOCATED
B7			PROHIBITED
B8			
B9	Sy_data_in_real<2><2>	INPUT	LOCATED
B10			
B11	Sy_data_in_real<2><0>	INPUT	LOCATED
B12	Sy_data_in_real<0><6>	INPUT	LOCATED
B13			
B14			
B15			
B16			
B17	clk	INPUT	LOCATED
B18			
C1			
C2			PROHIBITED
C3			PROHIBITED
C4			PROHIBITED
C5	Sy_data_in_real<1><0>	INPUT	LOCATED
C6	Sy_data_in_real<1><2>	INPUT	LOCATED
C7	GND		
C8	Sy_data_in_real<0><3>	INPUT	LOCATED

C9			
C10	Sy_data_in_real<2><5>	INPUT	LOCATED
C11	Sy_data_in_real<2><4>	INPUT	LOCATED
C12	GND		
C13			
C14	Wr_not<2>	INPUT	LOCATED
C15	M1		
C16	M2		
C17	Adr_Sy<2>	INPUT	LOCATED
C18			
D1			
D2			PROHIBITED
D3	VCC		
D4	GND		
D5	Sy_data_in_real<1><5>	INPUT	LOCATED
D6	Sy_data_in_real<0><5>	INPUT	LOCATED
D7			
D8	Sy_data_in_real<0><2>	INPUT	LOCATED
D9	GND		
D10	VCC		
D11	Sy_data_in_real<0><7>	INPUT	LOCATED
D12	Sy_data_in_real<1><7>	INPUT	LOCATED
D13			
D14			
D15	GND		
D16	VCC		
D17			
D18			
E1			
E2			
E3	Sy_data_in_real<0><0>	INPUT	LOCATED
E4			
E15			
E16			PROHIBITED
E17			PROHIBITED
E18			
F1			
F2			
F3			PROHIBITED
F4			
F15			
F16			
F17			
F18			
G1			PROHIBITED
G2			PROHIBITED
G3	GND		
G4			
G15	Sy_data_out<1>	OUTPUT	LOCATED
G16	GND		
G17			
G18			
H1			PROHIBITED
H2			
H3			
H4			
H15	Sy_data_out<3>	OUTPUT	LOCATED
H16			
H17	Sy_data_out<0>	OUTPUT	LOCATED
H18	Sy_data_out<4>	OUTPUT	LOCATED
J1			PROHIBITED

J2			PROHIBITED
J3			PROHIBITED
J4	VCC		
J15	VCC		
J16			PROHIBITED
J17	Sy_data_out<2>	OUTPUT	LOCATED
J18	Sy_data_out<16>	OUTPUT	LOCATED
K1			PROHIBITED
K2			PROHIBITED
K3			PROHIBITED
K4	GND		
K15	GND		
K16	Sy_data_out<17>	OUTPUT	LOCATED
K17	reset	INPUT	LOCATED
K18	downsampl_del	INPUT	LOCATED
L1			PROHIBITED
L2			
L3			
L4			
L15	Sy_data_out<14>	OUTPUT	LOCATED
L16	Sy_data_out<6>	OUTPUT	LOCATED
L17	Sy_data_out<5>	OUTPUT	LOCATED
L18	Sy_data_out<15>	OUTPUT	LOCATED
M1			PROHIBITED
M2			PROHIBITED
M3	GND		
M4			
M15	Sy_data_in_imag<0><0>	INPUT	LOCATED
M16	GND		
M17	Sy_data_in_imag<0><1>	INPUT	LOCATED
M18	Sy_data_in_imag<2><7>	INPUT	LOCATED
N1			
N2			
N3			PROHIBITED
N4			
N15	Sy_data_in_imag<0><6>	INPUT	LOCATED
N16	Sy_data_in_imag<1><3>	INPUT	LOCATED
N17	Sy_data_in_imag<0><7>	INPUT	LOCATED
N18	Sy_data_in_imag<0><4>	INPUT	LOCATED
P1			
P2			
P3	Wr_not<1>	INPUT	LOCATED
P4			
P15	Sy_data_in_imag<0><5>	INPUT	LOCATED
P16	Sy_data_in_imag<2><1>	INPUT	LOCATED
P17	Sy_data_in_imag<1><7>	INPUT	LOCATED
P18	Sy_data_in_imag<0><2>	INPUT	LOCATED
R1			
R2			
R3	GND		
R4	VCC		
R5			
R6			
R7	Sy_data_out<21>	OUTPUT	LOCATED
R8	Sy_data_out<10>	OUTPUT	LOCATED
R9	GND		
R10	VCC		
R11			
R12			
R13	Sy_data_in_imag<2><2>	INPUT	LOCATED
R14	Sy_data_in_imag<2><3>	INPUT	LOCATED

R15	VCC		
R16	GND		
R17	Sy_data_in_imag<1><1>	INPUT	LOCATED
R18	Sy_data_in_imag<0><3>	INPUT	LOCATED
T1			
T2			PROHIBITED
T3			PROHIBITED
T4			PROHIBITED
T5	Adr_Sy<1>	INPUT	LOCATED
T6			
T7	GND		
T8	Sy_data_out<7>	OUTPUT	LOCATED
T9			PROHIBITED
T10			
T11			
T12	GND		
T13	Sy_data_in_imag<2><4>	INPUT	LOCATED
T14	Sy_data_in_imag<1><4>	INPUT	LOCATED
T15			PROHIBITED
T16	Adr_Sy<0>	INPUT	LOCATED
T17	Sy_data_in_imag<1><2>	INPUT	LOCATED
T18	Sy_data_in_imag<1><5>	INPUT	LOCATED
TDO	TDO		
U1			PROHIBITED
U2	TDO		
U3			PROHIBITED
U4	Sy_rdy_in	INPUT	LOCATED
U5			
U6	Sy_data_out<20>	OUTPUT	LOCATED
U7	Sy_data_out<18>	OUTPUT	LOCATED
U8	Sy_data_out<8>	OUTPUT	LOCATED
U9			PROHIBITED
U10			PROHIBITED
U11	Sy_data_out<12>	OUTPUT	LOCATED
U12			PROHIBITED
U13			
U14	Sy_data_in_imag<2><5>	INPUT	LOCATED
U15	Sy_data_in_imag<1><6>	INPUT	LOCATED
U16	Sy_data_in_imag<1><0>	INPUT	LOCATED
U17	DONE		
U18	Wr_not<0>	INPUT	LOCATED
V1	CCLK		
V2			PROHIBITED
V3			PROHIBITED
V4			
V5			
V6	Sy_data_out<19>	OUTPUT	LOCATED
V7			PROHIBITED
V8	Sy_data_out<13>	OUTPUT	LOCATED
V9	Sy_data_out<11>	OUTPUT	LOCATED
V10			
V11	Sy_data_out<9>	OUTPUT	LOCATED
V12			PROHIBITED
V13			
V14			
V15	Sy_data_in_imag<2><6>	INPUT	LOCATED
V16	Sy_data_in_imag<2><0>	INPUT	LOCATED
V17			PROHIBITED
V18	/PROG		

```
?inout constraints listing
Fhese constraints are in PCF grammar format
and may be cut and pasted into the PCF file
after the "SCHEMATIC END ;" statement to
preserve this pinout for future design iterations.

MP "Adr_Sy<0>" LOCATE = SITE "T16" ;
MP "Adr_Sy<1>" LOCATE = SITE "T5" ;
MP "Adr_Sy<2>" LOCATE = SITE "C17" ;
MP "Sy_data_in_imag<0><0>" LOCATE = SITE "M15" ;
MP "Sy_data_in_imag<0><1>" LOCATE = SITE "M17" ;
MP "Sy_data_in_imag<0><2>" LOCATE = SITE "P18" ;
MP "Sy_data_in_imag<0><3>" LOCATE = SITE "R18" ;
MP "Sy_data_in_imag<0><4>" LOCATE = SITE "N18" ;
MP "Sy_data_in_imag<0><5>" LOCATE = SITE "P15" ;
MP "Sy_data_in_imag<0><6>" LOCATE = SITE "N15" ;
MP "Sy_data_in_imag<0><7>" LOCATE = SITE "N17" ;
MP "Sy_data_in_imag<1><0>" LOCATE = SITE "U16" ;
MP "Sy_data_in_imag<1><1>" LOCATE = SITE "R17" ;
MP "Sy_data_in_imag<1><2>" LOCATE = SITE "T17" ;
MP "Sy_data_in_imag<1><3>" LOCATE = SITE "N16" ;
MP "Sy_data_in_imag<1><4>" LOCATE = SITE "T14" ;
MP "Sy_data_in_imag<1><5>" LOCATE = SITE "T18" ;
MP "Sy_data_in_imag<1><6>" LOCATE = SITE "U15" ;
MP "Sy_data_in_imag<1><7>" LOCATE = SITE "P17" ;
MP "Sy_data_in_imag<2><0>" LOCATE = SITE "V16" ;
MP "Sy_data_in_imag<2><1>" LOCATE = SITE "P16" ;
MP "Sy_data_in_imag<2><2>" LOCATE = SITE "R13" ;
MP "Sy_data_in_imag<2><3>" LOCATE = SITE "R14" ;
MP "Sy_data_in_imag<2><4>" LOCATE = SITE "T13" ;
MP "Sy_data_in_imag<2><5>" LOCATE = SITE "U14" ;
MP "Sy_data_in_imag<2><6>" LOCATE = SITE "V15" ;
MP "Sy_data_in_imag<2><7>" LOCATE = SITE "M18" ;
MP "Sy_data_in_real<0><0>" LOCATE = SITE "E3" ;
MP "Sy_data_in_real<0><1>" LOCATE = SITE "B1" ;
MP "Sy_data_in_real<0><2>" LOCATE = SITE "D8" ;
MP "Sy_data_in_real<0><3>" LOCATE = SITE "C8" ;
MP "Sy_data_in_real<0><4>" LOCATE = SITE "B6" ;
MP "Sy_data_in_real<0><5>" LOCATE = SITE "D6" ;
MP "Sy_data_in_real<0><6>" LOCATE = SITE "B12" ;
MP "Sy_data_in_real<0><7>" LOCATE = SITE "D11" ;
MP "Sy_data_in_real<1><0>" LOCATE = SITE "C5" ;
MP "Sy_data_in_real<1><1>" LOCATE = SITE "B3" ;
MP "Sy_data_in_real<1><2>" LOCATE = SITE "C6" ;
MP "Sy_data_in_real<1><3>" LOCATE = SITE "A5" ;
MP "Sy_data_in_real<1><4>" LOCATE = SITE "B5" ;
MP "Sy_data_in_real<1><5>" LOCATE = SITE "D5" ;
MP "Sy_data_in_real<1><6>" LOCATE = SITE "A12" ;
MP "Sy_data_in_real<1><7>" LOCATE = SITE "D12" ;
MP "Sy_data_in_real<2><0>" LOCATE = SITE "B11" ;
MP "Sy_data_in_real<2><1>" LOCATE = SITE "A8" ;
MP "Sy_data_in_real<2><2>" LOCATE = SITE "B9" ;
MP "Sy_data_in_real<2><3>" LOCATE = SITE "A7" ;
MP "Sy_data_in_real<2><4>" LOCATE = SITE "C11" ;
MP "Sy_data_in_real<2><5>" LOCATE = SITE "C10" ;
MP "Sy_data_in_real<2><6>" LOCATE = SITE "A3" ;
MP "Sy_data_in_real<2><7>" LOCATE = SITE "A4" ;
MP "Sy_data_out<0>" LOCATE = SITE "H17" ;
MP "Sy_data_out<10>" LOCATE = SITE "R8" ;
MP "Sy_data_out<11>" LOCATE = SITE "V9" ;
```

```
COMP "Sy_data_out<12>" LOCATE = SITE "U11" ;
COMP "Sy_data_out<13>" LOCATE = SITE "V8" ;
COMP "Sy_data_out<14>" LOCATE = SITE "L15" ;
COMP "Sy_data_out<15>" LOCATE = SITE "L18" ;
COMP "Sy_data_out<16>" LOCATE = SITE "J18" ;
COMP "Sy_data_out<17>" LOCATE = SITE "K16" ;
COMP "Sy_data_out<18>" LOCATE = SITE "U7" ;
COMP "Sy_data_out<19>" LOCATE = SITE "V6" ;
COMP "Sy_data_out<1>" LOCATE = SITE "G15" ;
COMP "Sy_data_out<20>" LOCATE = SITE "U6" ;
COMP "Sy_data_out<21>" LOCATE = SITE "R7" ;
COMP "Sy_data_out<2>" LOCATE = SITE "J17" ;
COMP "Sy_data_out<3>" LOCATE = SITE "H15" ;
COMP "Sy_data_out<4>" LOCATE = SITE "H18" ;
COMP "Sy_data_out<5>" LOCATE = SITE "L17" ;
COMP "Sy_data_out<6>" LOCATE = SITE "L16" ;
COMP "Sy_data_out<7>" LOCATE = SITE "T8" ;
COMP "Sy_data_out<8>" LOCATE = SITE "U8" ;
COMP "Sy_data_out<9>" LOCATE = SITE "V11" ;
COMP "Sy_rdy_in" LOCATE = SITE "U4" ;
COMP "Wr_not<0>" LOCATE = SITE "U18" ;
COMP "Wr_not<1>" LOCATE = SITE "P3" ;
COMP "Wr_not<2>" LOCATE = SITE "C14" ;
COMP "clk" LOCATE = SITE "B17" ;
COMP "downsampl_del" LOCATE = SITE "K18" ;
COMP "reset" LOCATE = SITE "K17" ;
#
```